21世纪高等学校规划教材 | 软件工程





软件工程

韩利凯 主编
高寅生 杨全 副主编

清华大学出版社

21 世纪高等学校规划教材·软件工程

软件工程

韩利凯　主编
高寅生　杨　全　副主编

清华大学出版社
北　京

内 容 简 介

本书针对高校计算机相关专业软件工程课程的需要而编写，书中系统地介绍了软件工程的基础知识与应用技术，内容包括软件工程的基本概念和基本知识，软件生命周期与软件开发的各种模型，软件立项与合同，软件需求分析的概念、方法和工具，软件策划的规模、费用和资源的估计方法，软件建模的思想及三个模型分析，软件设计概论和设计方法，软件测试方法，软件实施及维护的方法，软件管理。本书在内容上注重科学性、先进性，强调实践性，提供了丰富的软件开发实例和素材，反映了软件工程的最新发展技术。

本书内容全面、深入浅出、理论和实践相结合，通过对本书的学习读者能够较好地掌握软件工程的基本知识和基本技术。本书可作为高等院校计算机科学与技术、软件工程等专业的软件工程课程教材，也可作为软件工程培训班教材或者软件开发及软件管理人员的自学参考书。

图书在版编目(CIP)数据

软件工程/韩利凯主编. --北京：清华大学出版社，2013
21 世纪高等学校规划教材 · 软件工程
ISBN 978-7-302-32785-1

Ⅰ. ①软…　Ⅱ. ①韩…　Ⅲ. ①软件工程－高等学校－教材　Ⅳ. ①TP311.5

中国版本图书馆 CIP 数据核字(2013)第 136197 号

责任编辑：郑寅堃　王冰飞
封面设计：
责任校对：焦丽丽
责任印制：

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编**：100084
社 总 机：010-62770175　　**邮　　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课件下载：http://www.tup.com.cn，010-62795954
印 刷 者：
装 订 者：
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张**：19　　**字　　数**：460 千字
版　　次：2013 年 9 月第 1 版　　**印　　次**：2013 年 9 月第 1 次印刷
印　　数：1～　000
定　　价：　.00 元

产品编号：049683-01

出版说明

随着我国改革开放的进一步深化，高等教育也得到了快速发展，各地高校紧密结合地方经济建设发展需要，科学运用市场调节机制，加大了使用信息科学等现代科学技术提升、改造传统学科专业的投入力度，通过教育改革合理调整和配置了教育资源，优化了传统学科专业，积极为地方经济建设输送人才，为我国经济社会的快速、健康和可持续发展以及高等教育自身的改革发展做出了巨大贡献。但是，高等教育质量还需要进一步提高以适应经济社会发展的需要，不少高校的专业设置和结构不尽合理，教师队伍整体素质亟待提高，人才培养模式、教学内容和方法需要进一步转变，学生的实践能力和创新精神亟待加强。

教育部一直十分重视高等教育质量工作。2007 年 1 月，教育部下发了《关于实施高等学校本科教学质量与教学改革工程的意见》，计划实施"高等学校本科教学质量与教学改革工程（简称'质量工程'）"，通过专业结构调整、课程教材建设、实践教学改革、教学团队建设等多项内容，进一步深化高等学校教学改革，提高人才培养的能力和水平，更好地满足经济社会发展对高素质人才的需要。在贯彻和落实教育部"质量工程"的过程中，各地高校发挥师资力量强、办学经验丰富、教学资源充裕等优势，对其特色专业及特色课程（群）加以规划、整理和总结，更新教学内容、改革课程体系，建设了一大批内容新、体系新、方法新、手段新的特色课程。在此基础上，经教育部相关教学指导委员会专家的指导和建议，清华大学出版社在多个领域精选各高校的特色课程，分别规划出版系列教材，以配合"质量工程"的实施，满足各高校教学质量和教学改革的需要。

为了深入贯彻落实教育部《关于加强高等学校本科教学工作，提高教学质量的若干意见》精神，紧密配合教育部已经启动的"高等学校教学质量与教学改革工程精品课程建设工作"，在有关专家、教授的倡议和有关部门的大力支持下，我们组织并成立了"清华大学出版社教材编审委员会"（以下简称"编委会"），旨在配合教育部制定精品课程教材的出版规划，讨论并实施精品课程教材的编写与出版工作。"编委会"成员皆来自全国各类高等学校教学与科研第一线的骨干教师，其中许多教师为各校相关院、系主管教学的院长或系主任。

按照教育部的要求，"编委会"一致认为，精品课程的建设工作从开始就要坚持高标准、严要求，处于一个比较高的起点上；精品课程教材应该能够反映各高校教学改革与课程建设的需要，要有特色风格、有创新性（新体系、新内容、新手段、新思路，教材的内容体系有较高的科学创新、技术创新和理念创新的含量）、先进性（对原有的学科体系有实质性的改革和发展，顺应并符合 21 世纪教学发展的规律，代表并引领课程发展的趋势和方向）、示范性（教材所体现的课程体系具有较广泛的辐射性和示范性）和一定的前瞻性。教材由个人申报或各校推荐（通过所在高校的"编委会"成员推荐），经"编委会"认真评审，最后由清华大学出版

社审定出版。

目前,针对计算机类和电子信息类相关专业成立了两个“编委会”,即“清华大学出版社计算机教材编审委员会”和“清华大学出版社电子信息教材编审委员会”。推出的特色精品教材包括:

(1) 21 世纪高等学校规划教材·计算机应用——高等学校各类专业,特别是非计算机专业的计算机应用类教材。

(2) 21 世纪高等学校规划教材·计算机科学与技术——高等学校计算机相关专业的教材。

(3) 21 世纪高等学校规划教材·电子信息——高等学校电子信息相关专业的教材。

(4) 21 世纪高等学校规划教材·软件工程——高等学校软件工程相关专业的教材。

(5) 21 世纪高等学校规划教材·信息管理与信息系统。

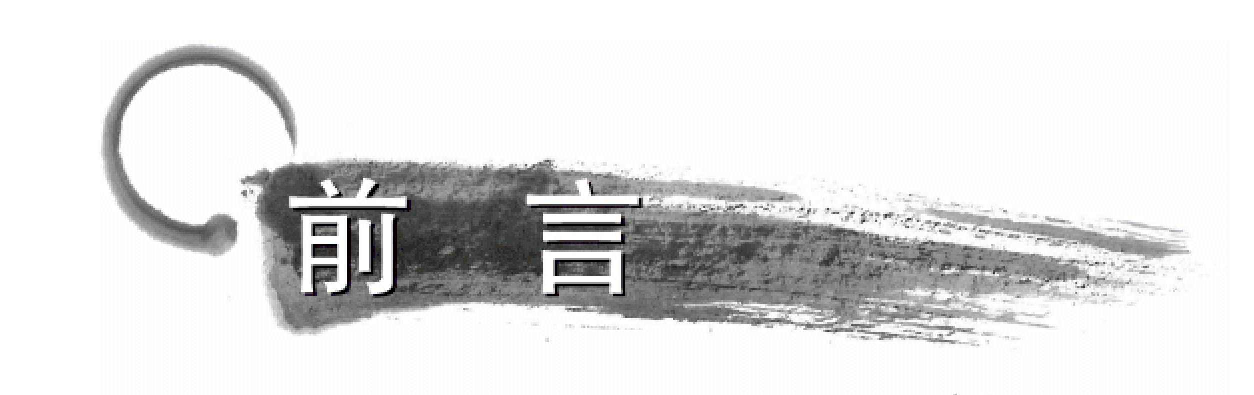
(6) 21 世纪高等学校规划教材·财经管理与应用。

(7) 21 世纪高等学校规划教材·电子商务。

(8) 21 世纪高等学校规划教材·物联网。

清华大学出版社经过三十多年的努力,在教材尤其是计算机和电子信息类专业教材出版方面树立了权威品牌,为我国的高等教育事业做出了重要贡献。清华版教材形成了技术准确、内容严谨的独特风格,这种风格将延续并反映在特色精品教材的建设中。

清华大学出版社教材编审委员会
联系人:魏江江
E-mail:weijj@tup.tsinghua.edu.cn



前　言

软件工程是将软件开发、维护和软件管理的理论应用于实践的一门工程科学，是计算机科学技术及相关专业的专业基础课。软件工程的概念主要是针对20世纪60年代发生的“软件危机”而提出的，自提出“软件工程”的概念以来，在四十多年的时间里，软件工程在软件开发模型、方法以及支持工具的研究方面都取得了长足的进步，在结构化程序设计、结构化方法、软件项目管理方法和工具研究等方面取得了大量研究成果，软件工程的应用水平已成为软件发展的关键因素。

随着计算机网络的发展和信息化水平的不断提高，大型软件系统的规模和复杂度与日俱增，软件技术面临着许多新的挑战。大型复杂软件的开发是一项特殊的工程，不仅与传统工程一样，需要按照工程化的方法去组织管理软件的开发，而且软件开发更具特殊性和复杂性。因此软件工程已经成为计算机科学与技术学科的重要基础学科。

在信息化趋势的引导下，随着各种大型软件的规模和复杂性的大幅度提升，软件质量可靠性的问题变得日益突出。几乎每个软件开发企业的软件产品在开发过程中都需要用到软件工程的相关知识，而这些工作必须依靠拥有娴熟技术的专业软件管理人员和开发人员来完成。软件项目管理人员就是这样的一个重要角色。但是目前在国内，在大多数软件企业中，软件开发管理人员的工程管理水平远远落后于国外先进水平。

随着高等院校计算机相关专业软件工程课程的开设，软件工程教学在高校中已经广泛开展，使软件工程课程的教学得到进一步改善。由此，对软件过程课程教材的需求也与日俱增，特别是理论和实践相结合的优秀教材。

本书的编写特色有两个：

(1) 理论和实践相结合，在理论基础上拓展实践技能。本书的前面部分详细介绍了软件工程理论知识，中间部分穿插讲述实际软件项目的工程案例，将理论知识加以应用，以提高本书的实践性。

(2) 注重知识的多元性。全书贯穿了软件工程、软件质量管理和软件项目管理等知识，让读者体会到软件工程在整个软件生命周期中的地位和作用。

本书由“软件危机”引出软件工程概念，逐步分析了软件工程的发展状况：第1章主要介绍软件工程的基本概念、应用前景和相关理论；第2章详细讲解软件生命周期与开发模型，以及各种模型之间的关系；第3章主要讲述软件立项、签订合同、软件招投标等的方法；第4章详细描述软件需求分析的基本概念、需求分析的方法、工具盒过程管理等；第5章详细介绍软件策划概论，规模估计、费用与资源估计的方法；第6章详细讲述软件建模思想以及三个模型建模实例分析；第7章具体介绍软件设计概论和原理，详细讲述面向过程设计、面向对象设计、面向元数据设计的方法；第8章介绍软件测试概论、软件测试模型，黑盒测试、白盒测试和灰盒测试的方法；第9章重点介绍软件实施与维护的基本知识以及软件维

护方法的比较；第 10 章讲述软件管理的相关知识，包括改进 CMMI 模型、软件配置管理、软件质量保证和软件项目管理的相关知识。

本书由韩利凯主编，高寅生、杨全任副主编。各章节编写安排如下：第 1 章、第 2 章：车鹏飞；第 3 章、第 4 章：刘鹰；第 5 章：徐东升；第 6 章：杨全；第 7 章、第 8 章：袁佳乐；第 9 章：任强；第 10 章：杨全。本书在编写过程中得到了多方面的帮助、指导和支持。感谢西安文理学院软件学院的各位领导和各位老师。

作　者

2013 年 6 月

目录

第1章 软件工程概论

在计算机软件技术发展初期,程序与软件的概念基本等同,开发软件也被认为是编写程序。随着计算机及软件技术应用的不断发展与推广扩大,软件逐渐发展成为独立的产业、产品。软件工程就是为适应软件的产业化发展需要,而逐步发展起来的,成为计算机和信息产业的重要支柱。本章主要介绍软件和软件工程的相关概念,概要介绍软件开发的过程及方法。

1.1 软件的定义

1.1.1 软件的概念及特点

1. 软件的概念

“软件(Software)”一词是在 20 世纪 60 年代出现的。人们普遍认为,软件是计算机系统中的一个重要组成部分,与计算机硬件系统共同组成计算机系统。早期的软件系统主要用于一些科研机构的科学工程计算,任务单一、应用领域很小。因此早期的软件开发主要侧重于程序编写,程序也成为早期软件的代名词。但随着软件系统功能的扩展,软件应用范围与规模的扩大,软件系统越来越复杂了。

就当前来讲,软件对于我们并不陌生,但是软件的含义并不明确,还有不少人错误地认为“软件就是程序,软件开发就是编写程序”。今天的软件系统已经具有了更多内容,包含计算机程序、用于设置程序正常工作的配置文件、描述软件构造的系统文档、指导软件使用的相关文档等。

人们普遍认可的软件的传统定义为:软件是计算机系统中与硬件相互依存的一部分,软件包括程序、数据及其相关文档的完整集合。其中,程序是按事先设计的功能和性能要求执行的指令序列,数据是程序能够正确地处理信息的数据结构,文档是与程序开发、维护和使用有关的图文资料。

软件是用户与硬件之间的接口,使用计算机就必须有软件,用户通过软件与计算机系统进行交互。软件在计算机系统的作用就是指挥和管理,计算机系统是否工作、怎么工作都是由软件系统决定的。

2. 软件的特点

软件是计算机系统中的逻辑成分,相对于硬件有形的物理特征,软件是无形的抽象概

念。例如，程序是操纵计算机工作的指令的集合，但它并不能以一种特殊的物理形态独立存在，必须通过物理存储介质，如通过硬盘等才能保存，当需要程序工作时将它调入到计算机内存中，并且通过计算机的中央处理器才能工作。

软件虽然也是由人工编写的产品，但与普通的人工制品有着很大差异，具有许多突出的特点。

1）形态特性

软件是无形的、不可见的逻辑实体，不是具体的物理实体，具有抽象性。

2）智能特性

软件是一系列的智力劳动的产品，软件的开发过程凝聚了大量的脑力劳动，它本身也体现了知识、实践经验和人类的智慧，具有一定的智能。软件可以帮助人类解决复杂的数学、分析、决策、管理等问题。

3）开发特性

软件是开发出来的，不是制造出来的。与传统的制造业或硬件开发不同，软件没有明显的制造过程，因此软件的质量完全取决于软件的开发过程。在开发过程中通过智力劳动和有效管理，可将知识和技术转化为软件产品。尽管现在已经有了一些软件开发工具用于辅助开发，但还是无法实现自动化生产。况且软件产品大多是根据功能需求进行定制开发的个性化产品。软件产品一旦开发完成，在后期就可以大量复制相同内容生成副本，再将副本进行应用。

4）质量特性

软件产品的质量在开发中很难控制，主要原因有以下几个方面：

（1）软件的需求在软件开发初期经常是不确切的，用户的需求和技术人员的理解有一定的差异性，而且需求在开发过程中往往会有所变更，使得软件的质量控制很难有个参照标准。

（2）软件的测试工作和测试技术有客观的局限性，软件的测试工作一方面是开发过程中的一个重要环节，另一方面任何测试只能在极大数量的应用实例以及应用情况中，选择有限的用例进行测试，无法做到对全部或者大多数实例进行测试工作，也就无法得到完全没有缺陷的软件产品。

（3）软件产品在长期使用中可能会出现无法预知的问题。

5）管理特性

软件开发的成功与否，不仅仅取决于技术层面，开发过程的管理是更为重要的。这种管理主要是对大规模知识型技术人员的智力劳动的管理，包括必要的培训、指导、激励、工作规范、过程以及进度的监控，此外还有沟通、协调等。

6）维护特性

任何实际使用中的产品都需要维护，但是软件产品的维护与其他的产品维护有很大的差异。传统产品的维修或维护基本都是由于使用造成了老化、磨损等问题，通过维修维护恢复功能或性能。软件产品出现问题一般并非使用造成，也不是使用时间长久造成的，软件维护往往是修正开发时遗留的、隐蔽的、没有发现的功能或性能缺陷。还有一种可能就是为了扩展和增加软件功能或性能，以及运行环境或特定运行需求的改进。

7）废弃特性

软件不会用坏，但可能被废弃。当软件运行的环境变化太大，或者用户提出了很多、很大的新需求时，如果对软件进行升级维护已经不现实或者不划算，该软件就存在失效与退化的问题。随着时间的推移，该软件将走到它的生命终点而被废弃。

1.1.2 软件的分类

计算机系统经常需要许多不同种类的软件协同工作，软件工程人员也会承担各种不同种类的软件开发、维护任务。不同种类的软件，对其进行开发和维护有着不同的要求和处理方法，现在还找不到一个统一的确定分类标准。因此可以从多个不同角度来划分软件的类别。

1. 按照功能划分

(1) 系统软件：是计算机系统的必要成分，它跟计算机硬件紧密配合，使计算机系统的各个部分协调、高效地工作。例如操作系统、数据库管理系统等。

(2) 支撑软件：是用来协助用户开发与维护软件系统的工具性软件，也称做工具软件。包括帮助程序人员开发软件产品的工具、帮助项目管理人员控制开发进度与质量的工具，支撑软件又可分为纵向支撑软件和横向支撑软件。纵向支撑软件是指支持软件生命周期各个阶段的软件工程活动所使用的软件工具，如需求分析工具、设计工具、编码工具、测试工具、维护工具等。横向支撑软件是指支持整个软件生命周期各个活动所使用的软件工具，如项目管理工具、配置管理工具等。20 世纪 90 年代中后期发展起来的软件开发环境以及后来开发的中间件是现代支撑软件的代表。

(3) 应用软件：是在系统软件的支持下，在特定领域内开发，最终为用户特定的目的提供服务的一类软件。例如商业数据处理软件、工程设计制图软件、办公自动化软件等。

(4) 可复用软件：最早的典型可复用的软件是各种标准函数库，通常是由计算机厂商提供的系统软件的一部分。后来可复用软件的范围扩展到算法之外，数据结构也可以复用。20 世纪 90 年代，可复用的范围从代码复用发展到体系结构的复用、开发过程的复用。面向对象开发方法的核心思想就是基于复用，为此建立了可复用的类库、应用程序库以及可复用的设计模式等。

2. 按照软件工作方式划分

(1) 实时处理软件：能够及时进行数据采集、反馈和迅速处理数据的软件，主要用于特殊设备的监控等场合。例如，汽车的自动驾驶控制软件系统。

(2) 分时处理系统：能够把计算机 CPU 工作时间轮流分配给多项数据处理任务的软件。例如，多任务操作系统。

(3) 交互式软件：能够实现人机对话式操作的软件。此类软件通常通过一定的操作界面接收用户输入的信息，并进行数据操作。这类软件在时间上没有严格的限定，但在操作上给了用户很大的灵活性。例如，商业数据处理软件系统的客户端程序。

(4) 批处理软件：能够把一组作业按照输入顺序或作业优先级等方式进行排队处理，并以成批加工的方式处理作业中的数据。例如，汇总报表打印程序。

1.1.3 软件的发展历程

自20世纪60年代中期以来,软件需求迅速增长,软件数量逐渐增多,逐渐发展成为一个专门的产业领域。经过几十年的发展,软件生产经历了三个发展阶段,即程序设计阶段、程序系统阶段和软件工程阶段。

1. 程序设计阶段(20世纪50年代)

程序设计阶段的软件是以个体手工的方式制作开发的,他们将机器语言、汇编语言作为工具直接面对机器编写程序代码。此时的硬件设备不仅价格昂贵,而且存储容量小、运行速度慢、功能可靠性差。尽管程序要完成的任务在今天看来是简单的,但由于受到计算机硬件设备条件的限制,程序设计也必须通过编程技巧来满足程序运行效率的要求。

此阶段的程序大多是开发者自用,软件还没有形成为产品。程序大多是为某个具体应用而专门编写的,程序功能单一,往往只能在某台专门的计算机上工作,很难将程序由一台设备移植到另一台设备。

2. 程序系统阶段(20世纪60年代)

程序系统阶段在软件技术上出现了高级程序语言,程序编写的效率得到了很大提高,一些更大规模的、具有综合功能的软件被开发出来。出现了操作系统,有效地改善了应用软件的工作环境,并且让应用软件具有了可移植性。随着计算机应用领域的不断扩大,软件需求也不断增长,软件规模越来越大,在这个时期出现了“软件作坊”,然后产生了一些具有一定通用性的软件产品。

“软件作坊”使得软件产品具有了工业化特征,这时的软件开发者已在使用系统的方法来设计、制作软件,而不是孤立地对待每个程序。但是“软件作坊”的组织机构比较松散,使得这个阶段软件开发还不能形成为工业流程。

这个时期软件开发的主要内容仍然是程序编写,软件开发的核心问题是技术问题,开发更多地是依赖于个人创作。从而忽视了用户的意图与需求,除了程序之外的其他文档、技术标准、软件的维护等问题也往往被忽视了。软件已经成为产品,但软件生产的方式则是跟产品并不适宜的作坊创作方式。终于随着软件规模的不断扩大,软件危机爆发了。

3. 软件工程阶段(20世纪70年代起)

大型软件的开发是一项工程性任务,采用个体或合作方式不仅效率低,产品可靠性差,而且很难完成。只有采用工程化的方法才能更有效地进行设计开发。1968年在前联邦德国召开的计算机国际会议上,人们专门针对软件危机问题进行了讨论,会议上正式提出并使用了“软件工程”术语。从而诞生了一门新兴的工程学科——软件工程,使软件的发展步入到了一个新的时代。

在软件开发的方法中,自20世纪70年代以来的四十年里,结构化的工程方法获得了广泛应用,成为了一种成熟的软件工程方法学。从20世纪90年代起,基于面向对象的工程方法开始应用于软件开发之中。软件产品的开发开始采用工程的原理、技术和方法进行实施,来适应软件产业化发展的需要。

这个阶段是软件产业高速发展的时期，以软件为特征的“智能”产品不断涌现。尤其是网络通信技术、数据库技术与多媒体技术的结合，彻底改变了软件系统的体系结构，它使得计算机的潜能获得了更大程度的释放。以计算机软件为核心的信息技术的高速发展，已经使得人们的生活方式与生活节奏发生了根本性的变化。

“软件工程”产生以后，人们希望能够解决“软件危机”问题，但是，软件危机现象并没有得到彻底排除。尤其是一些老的危机问题得到解决了，但接着又出现许多新的危机问题，于是不得不去寻找一些更有效、更先进的工程方法。正是软件危机问题的不断出现，才推动着软件工程方法学的不断发展。

1.1.4 软件危机

1. 软件危机的介绍

软件危机是指在计算机软件的开发和维护过程中所遇到的一系列严重问题。例如，软件的开发成本、进度和软件质量等。这些问题并不仅仅是不能正常运行的软件才具有的，实际上，几乎所有软件都不同程度地存在这些问题。概括来讲，软件危机包含两方面的内容：如何开发软件以满足人们对软件日益增长的需求；如何维护数量不断扩大的已有软件。

具体来讲，软件危机主要有以下一些典型表现。

1）对软件开发成本和进度的估计很不准确

软件开发组织制定的项目计划跟实际情况有很大差距，实际成本比预算成本可能高出很多。由于对开发工作和开发难度估计不足，研发进度计划无法按时完成，软件开发一再拖延，这种情况严重降低了软件开发组织的信誉。而如果为了赶进度和节约成本采用一些手段方法，又会损害软件产品的质量。

2）软件产品通常与用户的需求有差异，导致用户不满意

在开发过程中，软件开发人员和用户之间缺乏充分的信息交流。开发人员常常是对用户要求只有模糊的了解，有时对所要解决的问题都没有确切的认识，但却过早地匆忙着手编写程序，导致开发出来的软件不能真正满足用户的实际需要。

3）软件产品的质量靠不住

在软件开发过程中，没有建立起确实有效的质量保证体系。软件质量保证技术没有真正地、坚持不懈地应用到开发全过程，软件项目往往为了控制进度或成本，不惜降低软件质量标准，导致出现软件产品质量问题。

4）软件文档资料不完整或不准确

计算机软件不仅仅是程序，还应该有一系列的整套文档资料。这些文档资料是在软件项目开发过程中产生的，对软件项目非常重要。在软件开发过程中，软件开发的管理人员需要使用这些文档资料来管理和评价项目的进展情况，技术人员则需要使用它们进行准确的信息交流，用户也需要软件文档资料来认识和熟悉软件，以便于验收、安装、使用等。但由于软件项目管理的不规范，软件文档往往不完整，对软件的描述经常是不一致的，很难通过文档去监测软件开发过程中的质量。

5）软件产品经常很难维护

软件中的程序错误是非常难改正的，软件也很难适应新的硬件环境，也不能根据用户的

需要在原有软件中增加新的功能。软件往往是不可以重用的，一旦过时就不得不完全废弃。可重用的软件目前还是一个正在努力追求的目标。

6）软件生产率低下

软件开发生产率提高的速度，远远落后于计算机应用的迅速普及。软件的开发也经常跟不上计算机硬件的发展，以至于无法充分发挥利用计算机的巨大性能潜力。

2. 产生软件危机的原因

软件开发和维护的过程中存在着许多严重问题，一方面和软件本身的特点有关，另一方面也和软件开发及维护的技术、方法不正确或不合适有关。

通过对软件危机现象的研究发现，产生软件危机的原因主要体现在以下几个方面。

1）软件的不可见特性

软件不同于硬件，它是计算机系统中的逻辑部件，软件缺乏"可见性"。在写出程序并在计算机系统运行调试之前，软件并没有直观的表现。软件开发过程的进展与质量也难以评价和控制，如果软件中存在程序错误，就必须等到运行时才有可能发现。软件的不可见特性使得对软件项目的量化管理很难实施。

2）软件系统规模庞大

软件产品不同于一般程序，显著特点之一就是规模庞大，随着功能的增多，其规模和复杂程度就越来越大。例如，1968 年美国航空公司订票系统拥有三十万条指令，1973 年美国阿波罗计划的软件系统拥有一千万条指令。面对不断复杂的软件系统，其开发手段仍然需要依靠开发人员的个人创造与手工劳动。

3）软件生产工程化管理程度低

软件开发的最终成果是软件产品，在软件开发过程中引入工程化的管理是非常必要的。为了在预订时间内开发出规模庞大的软件，必须由许多人分工合作，保证每个人完成的工作能够被集成起来构成一个高质量的软件系统，这本身就是一个极端困难复杂的问题，必须有严格的、科学的、规范的管理机制。在软件分析和设计完成之后，才能开始软件的实现。但是，实际上软件的开发则往往是在分析、设计没有完成的情况下，就已经进入编码实现的阶段。致使软件项目管理混乱，严重影响软件项目成本、开发进度和软件质量。

4）对用户需求关注不充分

软件是为用户开发的，只有用户才真正清楚自己的需要。软件开发组织不熟悉用户的业务领域。技术人员所关注的仅仅是计算机技术，对用户的需求调查不充分，对用户需求进行研究的策略、手段也有缺失，导致用户的需求信息不能被充分反映，或被错误理解。由于没有对用户需求做大量深入细致的调查研究，以致软件需求设计不准确，最终使得开发的软件不能适应用户的应用需要。

5）对软件维护重视程度不够

软件开发缺乏统一的、科学的规范。在软件产品开发过程中，开发者很少考虑软件今后需要提供的维护工作。同时在其漫长的使用周期内，软件错误大多具有隐蔽性，许多年之后软件仍可能需要改错。此外，软件的工作环境也可能发生改变，用户也可能在软件运行几年以后，要求增加新的功能，这些都属于软件维护的问题。软件的可维护性也是衡量软件质量的一项重要指标，可维护性程度高，软件就越便于修正、改版和升级，从而具有更长的使用

寿命。

6）软件开发工具自动化程度低

现在软件开发的工具已经有了很大的进步，但仍然离不开技术人员的个人创造与手工编写，软件生产还是不可能像硬件设备生产的那样，达到高度的自动化。

1.2　软件工程

1.2.1　软件工程的定义

虽然软件工程概念的提出已经有四十多年了，但直到目前为止，并没有一个对软件工程概念的统一的定义。概括地说，软件工程是指导软件开发和维护的一门工程学科，以计算机科学理论和其他相关学科的理论为指导，采用工程化的概念、原理、技术和方法来开发与维护软件，把经过时间考验而证明正确的管理技术和当前能够运用的最佳技术方法结合起来，以低成本开发出高质量的软件并进行维护。

软件工程涉及软件生产的各个方面，包括工程过程、工程原则、技术方法与工具，以及工程项目管理等，为经济地、高效地开发高质量软件产品提供最有效的支持。

关于软件工程的定义，可以从多个不同的角度来列举几个。

1983 年国际权威机构 IEEE 在软件工程术语汇编中，给软件工程下的定义是：软件工程是开发、运行、维护和修复软件的系统方法。其中的“软件”被定义为：计算机程序、方法、规则、相关的文档资料，以及计算机程序运行时所需要的数据。

1993 年 IEEE 进一步给出了一个更全面、更具体的定义：“软件工程是：①把系统的、规范的、可度量的途径应用于软件开发、运行和维护过程，也就是把工程应用于软件；②研究①中提到的途径。”

Fairly 对软件工程的定义是：软件工程学是为了在成本限额以内按时完成开发和修改软件产品时，所需要的系统生产和维护技术及管理学科。

Fritz Baner 的定义为：软件工程是为了经济地获得可靠的且能在实际机器上有效运行的软件，而建立和使用的、完善的工程化原则。

1.2.2　软件工程技术介绍

软件工程作为软件开发与维护的工程方法学，它所具有的技术要素就是软件工程技术。软件工程技术主要有三个方面：软件工程方法、软件工具和软件工程过程。

1. 软件工程方法

软件工程方法是指完成软件开发与维护任务时，需要“如何做”的技术方法，涉及了软件开发、维护的整个过程中的任务，包括软件需求分析、软件结构设计、程序算法设计等诸多任务。它的方法主要体现在，使用图形或某种特殊描述的方式来表现这些任务中需要建立的软件系统模型，例如数据流模型、软件结构模型、对象模型、组件模型等。软件工程方法主要有结构化方法、JSD 方法和面向对象方法。

1）结构化方法

结构化方法是基于软件生命周期的传统的软件工程方法，从20世纪70年代产生以来，获得了卓有成效的软件开发应用。结构化方法是以软件功能为目标进行的软件构建过程，包括结构化分析、结构化设计、结构化实现和结构化维护等内容，能够很好地适应结构化编程工具，例如C、Pascal语言等。主要使用数据流模型来描述软件的数据加工过程，并可以通过数据流模型，由对软件的分析顺利过渡到对软件的结构设计。

2）JSD方法

JSD方法主要用在软件设计上，由法国科学家Jackson在1983年提出。它以软件中的数据结构为主要依据进行软件结构与程序算法的设计，是结构化软件设计方法的一种有效补充。尤其在以数据处理为主要内容的信息系统开发中，JSD方法具有突出的设计建模优势。

3）面向对象方法

面向对象方法是以软件问题域中的对象为基本依据进行软件系统模型设计，包括面向对象分析、面向对象设计、面向对象实现和面向对象维护等内容。面向对象分析与设计过程的核心内容是，确定问题域中的对象成分及其关系、建立软件系统对象模型。自20世纪80年代以来，已经提出了许多有关面向对象的方法，其中由Booch、Rumbaugh、Jacobson等人提出的一系列面向对象方法成为了主流，并被结合为统一建模语言（UML），成为了面向对象方法中的公认标准。面向对象方法能够最有效地适应面向对象编程工具，例如C++、Java等语言，特别适用于面向用户的交互式系统开发。

2. 软件工具

软件工具是一种具有自动化特征的软件支撑环境，作用是对软件工程方法的运用提供有效支持。

软件工具通常称做CASE，它是计算机辅助软件工程（Computer-Aided Software Engineering）的英文缩写。CASE工具覆盖面很广，包括分析建模、设计建模、源代码编辑生成、软件测试等方面。

用来支持软件分析和设计的CASE工具，称为高端CASE工具，如数据字典管理器、分析建模图形编辑器、软件结构设计器等。而用来支持软件实现和测试的CASE工具，被称为低端CASE工具，如程序编辑器、程序分析器、源程序调试器等。

可以把诸多独立的软件工具集成起来，形成集成的、一体化的CASE工作平台。这样的工作平台能够为软件开发提供更强有力的支持，平台中数据资源共享，界面风格、操作方式统一，很多通用操作能够作为模块被许多工具调用，多种工具产生的信息也可以互相引用。

3. 软件工程过程

为了实现软件产品生产完全自动化，除了有软件工具与软件工程方法外，还需要有合适的软件工程过程才能真正发挥作用。

软件工程过程是指开发机构在开发软件产品过程中，在软件工具的支持下，按照一定的软件工程方法所进行的一系列软件工程活动。这一系列的活动实际就是软件开发中开发机

构需要制定的工作步骤，它应该是科学的、合理的，否则将影响软件开发成本、进度与产品质量。

软件工程过程涉及软件产品开发中有哪些工作步骤，各个工作步骤具有怎样的工作特征，以及各个工作步骤需要产生一些什么结果等方面。

软件工程过程并不是绝对固定的，软件开发机构可以专门制定适合自身特点的软件工程过程。在实际开发中，软件产品不同，软件工程过程也可能会有所不同。绝大多数软件工程过程都具有以下四项基本活动。

(1) 软件定义：进行软件规格和使用限制的定义。

(2) 软件开发：根据软件规格定义制作出软件产品。

(3) 软件验证：确认软件能够满足用户提出的要求。

(4) 软件维护：修正软件缺陷，并能根据用户需求变化改进软件。

1.2.3　软件工程的基本原理

自从正式提出了“软件工程”这个术语以来，研究软件工程的专家们陆续提出了一百多条有关软件工程的准则。1983 年美国 TRW 公司的 B. W. Boehm 将这些软件工程的准则概括为著名的软件工程七条基本原理。

1. 按软件生命周期分阶段制定计划，并进行严格管理

一个软件从定义、开发、运行和维护，直到最终被废弃丢掉，要经历一个很长的期间，通常被称为软件生命周期。在软件开发与维护的漫长的时间中，需要完成许多性质各异的工作。应该把软件生命周期划分为若干阶段，并相应地制定切实可行的计划，然后严格按照计划对软件的开发与维护工作进行管理。不同层次的管理人员都必须严格按照计划认真尽职地管理软件的开发与维护工作，绝不能受客户或上级人员的影响而擅自违背预订计划。

2. 坚持进行阶段评审

软件的质量保证工作不能等到编码阶段结束之后再进行，原因有两个：第一个，大部分错误是在编码之前造成的，根据有关统计，设计错误占软件错误的 63%，编码错误仅占 37%；第二个，错误发现得越晚，为改正它所需要付出的代价就越大。因此，在每个阶段都应进行严格的评审，以便尽早发现在软件开发过程中的错误。

3. 实行严格的产品控制

在软件开发过程中不应随意改变需求，因为改变一项需求往往需要付出较高的代价。但是，由于外界环境的变化或软件工作范围的变化，在软件开发过程中改变需求又是难免的，只能依靠科学的产品控制技术来适应需求的变更。

4. 采用现代程序设计技术

自从提出软件工程的概念开始，开发者一直把主要精力用于研究各种新的程序设计技术，实践证明采用先进的技术不仅可以提高软件开发和维护的效率，而且可以提高软件的质量。

5. 具有明确的责任

软件产品不同于一般的工业产品，是看不见摸不着的逻辑产品。软件开发组织的工作进展情况可见性差、可控性差，难以准确度量，使得软件产品的开发过程比一般产品的开发过程更难以评价和管理。为了提高软件开发过程的可见性，更好地进行进度控制管理，应当根据软件开发项目的总目标及进度安排，规定开发者的责任和任务标准，使工作结果能够被清楚地审查。

6. 开发组织的人员尽量少而精

软件开发组织的组成人员应当有很高的素质，但是人数则不宜过多，提高人员的素质能促进软件开发产品质量的提高和开发效率的提高。高素质的开发人员可以明显减少软件中的错误，而且开发小组随着人数的增加，会因交流开发进度等情况和讨论遇到问题而造成额外的开销，因此应当保证软件开发组织的人员少而精。

7. 不断改进软件开发过程

软件开发过程是将软件工程的方法和开发技术、开发工具综合起来，达到合理、及时地进行计算机软件系统开发的目的。开发过程定义了方法使用的顺序、要求交付的文档资料、为保证开发质量和项目变化所需要的管理，以及软件开发各个阶段进行的评审工作。为保证软件的过程能适应软件开发技术的进步，必须不断改进软件工程过程，应当积极主动地采用新的软件开发技术，不断总结经验。

1.3 软件工程在软件行业中的作用

软件开发的项目一般都有多个层次的、不同分工的人员互相配合，在开发项目的各个部分以及各开发阶段之间都存在着许多联系和衔接问题。如何把这些错综复杂的关系协调好，需要有一系列统一的约束和规定。在软件开发项目取得阶段成果或最后完成时，需要进行阶段评审和验收测试。投入运行的软件，其维护工作中遇到的问题又与开发工作有着密切的关系。软件与管理工作则渗透到软件生命周期中的每一个环节。所有这些都要求提供统一的行动规范和衡量准则，使得各种工作都有章可循。

软件工程在软件开发中的应用可以带来以下一些益处：

（1）提高软件的可靠性、可维护性和可移植性，可以提高软件产品的质量。

（2）提高软件的生产率，提高软件人员的技术水平。

（3）提高软件人员之间的通信效率，减少差错和误解。

（4）有利于软件开发工作的标准化管理。

（5）有利于降低软件产品的成本和运行维护成本。

（6）有利于缩短软件的开发周期。

在软件行业的实际工作中，软件工程在开发中的作用主要表现在以下一些方面。

1. 定义软件项目成功的标准

在项目的开始，要对开发的软件项目是否成功有一个统一的认识标准。一般情况下，满足一个预定义的进度安排是唯一明显的成功因素，但是肯定还有其他的因素存在。比如：增加市场占有率，获得指定的销售量或销售额，达到特定用户的特定满意程度，淘汰一个高维护需求的遗留系统，取得一个特定的事务处理量并保证正确性，项目计划的目标定义，包括进度、成本和质量。

2. 定义项目的驱动、约束和自由程度

每个软件项目都需要平衡它的功能、人员、预算、进度和质量指标。我们把以上五个项目指标中的每一个方面，要么定义成一个约束，必须在这个约束中进行操作；要么定义成与项目成功对应的驱动；或者定义成通向成功的自由程度，可以在一个规定的范围内调整。

3. 定义产品发布标准

在项目早期，要决定用什么标准来确定产品是否准备好发布了。发布标准可基于：还存在多少个高优先级的缺陷、性能度量、特定功能完全可操作、或表明项目已经达到了它的目的的其他方面。不管你选择了什么标准，都应该是可实现的、可测量的、文档化的，并且与你的客户要求的“质量”一致。

4. 保持良好信息沟通

在项目开发中，需要和客户、管理人员以及开发人员保持良好的信息沟通，软件工程的管理思想和管理技术，可以帮助开发机构与客户以及其他相关人员进行及时的、良好的信息交流与沟通。

5. 制定项目计划

在软件开发工作中，有些人认为花时间写项目计划还不如将时间用来编写代码，但是实际的软件工程实践证明，制定项目计划是非常必要和必需的工作。制定计划的困难之处不是编写计划本身，而是制作这个计划所需要的思考、沟通、权衡、交流、问题以及解决方案。花费时间制定计划用来分析解决问题，会减少软件项目开发中出现的问题和意外。

6. 对项目的任务分解

采用软件工程的思想和管理方法，可以细致地、准确地把大任务分解成多个小任务，帮助开发机构更加精确地估计它们，有助于暴露出软件项目可能没有想到的一些工作活动，并且保证更加准确地跟踪开发状态。

7. 制定质量控制活动后的修改工作计划

几乎所有的质量控制活动，如测试和技术评审，都会发现项目的缺陷或提高其他方面性能的可能。按照软件工程的管理思想，可以把质量控制活动后的修改工作，制定成单独的任务计划。

8. 为过程改进安排时间

开发团队成员往往是淹没在他们当前的项目中，但是如果想提升开发小组的软件工程能力水平，就必须在过程改进方面投资一些时间。也就是从项目进度中留出一些时间，用来帮助开发团队改进更加成功的开发工作过程。

9. 管理项目的风险

软件工程的管理思想和方法可以帮助开发机构有效地识别和控制风险。在项目计划时花一些时间集体讨论可能的风险因素，评估它们的潜在危害，以及如何减轻或预防这些风险是十分必要的。

1.4 软件工程方法论

在对软件开发工作的研究中，软件工程进一步发展出软件过程的概念，软件工程需要研究“如何做”的软件方法，也要研究提供支持的软件工具和软件环境。软件过程则是研究如何综合使用软件方法和软件工具。为了更好地发展和改进软件工程技术，我们有必要从方法论的各个角度分析软件工程的方法、工具和过程，从而目标明确地改进软件工程中各个过程的思想、方法、模式和规则。

1. 软件工程的发展性

和其他技术方法一样，软件技术经历了由简单到复杂，由低级到高级，由以经验为基础到以科学为基础的历史过程。回顾软件技术的发展历史，可以帮助我们把握软件技术的本质特征，给我们一些有益的启示。

软件技术的发展可以分为三个阶段：程序设计阶段；程序系统阶段；软件工程阶段。经过几十年的发展，程序最根本的变化体现在观念认识方面：程序从依照个人意图创造的“艺术品”转变为被广大用户所接受的工程化产品；软件从满足设计者自己需求的生产方式转变到需要在市场中运作以满足广大用户的需要；软件开发工作的范围从只考虑程序的编写工作，扩展到涉及整个软件生命周期所包含的所有工作。

随着软件技术，尤其是面向对象技术(OO)的发展，软件工程提出了以下新的思想方法和设计原则。

(1) 抽象：抽取事物最基本的特性和行为，忽略非基本的细节。采用分层次抽象，自顶向下、逐层细化的办法控制软件开发过程的复杂性。

(2) 信息隐蔽：将模块设计成“黑箱”，将实现的细节隐藏在模块内部，不让模块的使用者直接访问，采用使用与实现分离的原则。

(3) 模块化：通过对象、类等模块化手段实现信息隐蔽和抽象，有助于表示复杂的系统。

(4) 局部化：在一个物理模块内集中逻辑上相互关联的计算机资源，保证模块之间具有松散的耦合，模块内部具有较强的内聚。这有助于控制问题的复杂性。

(5) 确定性：软件开发过程中所有概念的表达应是确定的、无歧义性的、规范的。

(6) 一致性：整个软件系统(包括程序、文档和数据)的各个模块应使用一致的概念、符

号和术语。

(7) 完备性：软件系统不应丢失任何重要成分，可以完全实现系统所要求功能的程度。为了保证系统的完备性，在软件过程中需要严格的技术评审。

(8) 可验证性：开发大型的软件系统需要对系统自顶向下、逐层分解。系统分解应遵循系统易于检查、测试、评审的原则，以确保系统的正确性。

软件技术的实践经验告诉我们，分解是解决复杂问题的有效途径。软件问题的分解一般都体现出整体和部分的叠加方式，一个复合体能够通过把原来分离的要素一步一步地合拢建立起来；反之，复合体的特征能够完全分解为各个分离要素的特征集合。所以在遵循上述设计原则下的问题分解能达到较好的效果。

软件技术的发展历史还启示我们，没有一成不变的方法，没有绝对适用的开发模式，随着软件技术的研究和发展，必然存在着新的、更实用的分析模型、设计思想和开发技术。

2. 软件工程的技术性

软件的定义包含了目前对软件技术外延的理解：所有的技术手段、途径和行为方式都是在程序、数据和文档的集合上的可操作的规则或模式。从方法论的角度分析软件问题，就必须考虑软件各要素作为技术方法所具有的特性。

(1) 实践性：软件工程的方法必须包含严格意义上的实践操作规则或模式，而不是限于理论和空谈。软件工程必须采用合适的符号体系，20 世纪 80 年代末以来，随着面向对象技术成为研究的热点，出现了几十种支持软件开发的面向对象方法。统一建模语言 UML(Unified Modeling Language)结合了 Booch、OMT 和 Jacobson 等方法的优点，统一了符号体系，并从其他的方法和工程实践中吸收了许多经过实际检验的概念和技术，成为对象管理集团(OMG)面向对象方法的标准。

(2) 社会性：软件工程的方法要关注社会因素，考虑机构、体制及管理方式等问题，甚至涉及人的观念和人们的心理。针对这一领域，美国卡耐基·梅隆大学软件工程研究所(SEI)提出了软件机构的能力成熟度模型 CMM，CMM 共分五级：初始级、可重复级、已定义级、已管理级和优化级。CMM 从人员、机构、体制及管理等众多角度为软件机构定义了可操作的分级实施和评估标准。

(3) 复杂性：软件本身是复杂的，软件的复杂性可能来自它所反映的实际问题的复杂性，也可能来自程序逻辑结构的复杂性。软件工程的方法要考虑到如何综合应用领域的知识，如何实现领域工程。

3. 软件工程的系统性

现代计算机的理论基础是图灵于 1937 年提出的图灵机模型以及相应的冯·诺依曼体系结构。是把问题转化为一步一步按规则执行的求解过程，各种计算机语言都是这种思想下的某种形式语言。软件开发的过程实质上就是，技术人员对客观世界问题域的形式化的过程。技术人员先建立问题的计算机语言表达，最后进行计算获得结果。由于计算机的实现过程和人类认识表达过程之间存在巨大鸿沟，技术人员往往把目光都集中在如何实现、如何编程的层面上。认识的偏差和思维的惯性导致对软件工程过程的支持不足。例如，传统的瀑布模型以项目的阶段评审和文档控制为手段对整个开发过程进行指导，但缺乏灵活性，

没有考虑到项目的评估和演进。

作为一种工程设计，必须对整个软件工程过程(Software Engineering Process)采用系统方法考虑其全过程。借鉴系统论"戴明循环法"的思路，软件工程过程包含四种基本的过程活动以及软件生命周期模型。

- P (Plan)：软件规格说明，规定软件的功能及其运行的限制。
- D (Do)：软件开发，产生满足规格说明的软件。
- C (Check)：软件确认，确认软件能够满足客户提出的要求。
- A (Action)：软件演进，为满足客户的变更要求，软件必须在使用的过程中演进。
- 演化模型：考虑到项目开发的初始阶段人们对软件的需求认识常常不够清晰，因而使得开发项目难以一次成功，出现返工再开发是在所难免的。因此，可以先做试验开发，探索可行性，弄清软件需求；然后在此基础上获得较为满意的软件产品。通常把第一次得到的试验性产品称为"原型"。
- 螺旋模型：对于复杂的大型软件，开发一个原型往往达不到要求。螺旋模型将瀑布模型与演化模型结合起来，并且加入两种模型均忽略了的风险分析。螺旋模型沿着"戴明循环法"的循环螺线旋转，沿螺线自内向外每旋转一圈便开发出更为完善的一个新的软件版本。
- 喷泉模型：喷泉模型对软件复用和生命周期中多项开发活动的集成提供了支持，主要支持面向对象的开发方法。系统某个部分常常重复工作多次，相关功能在每次迭代中随之加入演进的系统。开发活动，即分析、设计和编码之间不存在明显的边界。
- 智能模型：基于知识的软件开发模型，综合了上述若干模型，并把专家系统结合在一起。该模型应用基于规则的系统，采用归约和推理机制，帮助软件人员完成开发工作，并使维护在系统规格说明一级进行。

演化模型、螺旋模型、喷泉模型、智能模型的进步之处在于它们都考虑了人们认识表达过程的反复特性，借鉴了系统论的系统方法，较好地支持了项目的评估和演进，并针对应用的特点组织了软件生命周期的各个环节。

1.5 软件工程实践论

软件产品的规模和复杂度不断增加，传统程序方式的开发逐步被以项目为单位的方式所取代。软件项目发展的过程中，软件工程技术的进步一直是产业发展的动力。而软件工程又包含了多个过程模块，大致分为前期、开发、运作和维护几大部分。这就会涉及软件项目的管理、软件测试和软件维护。

1.5.1 软件项目管理

所谓项目，就是在特定条下、规定时间内、满足一系列特定目标的工作的总称。项目具有一次性、独特性、目标的确定性、组织的临时性和开放性等属性。项目管理是指"在项目中运用专门的知识、技能、工具和方法，使项目实现相关人的需要"，这不仅仅强调了使用专门的知识和技能，还强调了各参与人的重要性。

1. 软件项目管理的特殊性

软件项目管理和其他的项目管理相比有相当的特殊性。软件的整个过程都是设计过程，不需要大量的物质资源(除人力资源)，开发的产品以程序代码、文档为主，并没有物质成果；软件是纯知识产品，其开发进度和质量很难估量。软件系统的复杂性是导致开发过程中各种风险难以预见和控制的因素。

2. 软件项目管理的内容

软件项目管理是一种科学的管理手段，它是为了使软件项目能够按照预定的要求完成，对成本、人员、进度、质量等进行分析和管理的任务。对人员的组织与管理、软件度量、项目计划、风险管理、质量保证、过程能力配置管理等几个方面贯穿、交织于整个软件开发过程中。从软件工程的角度讲，软件开发主要分为需求分析阶段、概要设计阶段、详细设计阶段、编码阶段、测试阶段、安装及维护阶段六个阶段。不管是何种开发都会涉及这六个阶段。从用户的角度来看，软件项目的生命周期应该包括项目前期的论证工作、项目计划、软件开发、运行、维护。可见，软件项目管理的范围不包括传统的软件开发过程，应该包括开发前的准备工作以及运行维护工作。

3. 软件项目管理的关键因素

1）人力配置

人是决定项目成败的关键，也是影响软件质量的关键。因此软件项目管理应该以人为本，有效管理人力资源，合理配置人力资源。应该合理搭配，充分发挥每位成员的技术专长；培养团结的团队精神，规范良好的职业道德；建立健全制度，落实责任，营造良好的团队协作环境。

2）沟通管理

开发软件是知识性极强的工作，对人的依赖性远胜于其他行业，加强人员间的沟通，进行有效地沟通管理是软件项目成功一个要素。要想科学地组织、控制项目的实施过程，就必须进行信息沟通。

3）可靠的软件需求

软件需求是软件项目的前提，需求范围不确定，项目开发就会失去方向，项目最终以失败告终。软件需求应当是清楚、完整、详细、可实现和可测试的需求，并且项目有关的人员一致同意，开发人员应该反复和用户进行沟通，明确用户并发掘用户的潜需求，最大限度地满足用户的目标，开发出用户最理想的产品。

4）周密可行的计划

软件项目实施过程中，还必须有一个周密可行的项目计划。软件项目计划的目的是为完成软件工程和管理软件项目而制定的合理的计划，要想成功进行项目管理，就要对软件项目计划高度重视、周密制定、严格执行。只有严格进行计划才能使项目管理得以成功实施。

5）完备的文档资料

软件项目的文档作为软件产品的主要形式之一，是软件人员的劳动成果，在整个软件生命周期中占据重要位置。健全完备的文档资料便于软件的测试与后期的维护工作、对人员

的培训和项目的再开发发挥着重要作用,健全完备的文档资料也是软件项目成功的重要因素,在项目管理过程中应该得到高度重视。

6)严格的风险管理

软件项目的管理是存在风险的,我们应该提前有所防范,以最大限度地减少风险发生的几率。建立风险项目检查表是进行风险识别的有效地方法,检查表主要涉及产品规模、过程、技术、开发环境等风险检查。同时要依据风险描述、风险概率和风险影响三个要素对风险进行评价。

1.5.2 软件测试

软件测试是一个知识密集型的活动,测试人员都属于知识工作者,他们测试相关的知识、技巧、经验和灵感,在测试过程中有着重要的作用,测试人员如果没有丰富的测试经验与测试技巧,将无法保证测试的质量。

1. 国内外研究现状

目前国内外对在软件测试领域内实施知识管理的相关研究很少,迄今为止并没有找到在软件测试中实施知识管理的实例。国内知识管理的研究起步晚,同时专门从事软件测试的企业又很少,在测试领域内实施知识管理的需求刚刚出现。

此外,国外在与软件测试最相近的软件工程领域对知识管理也有了比较深入的相关研究,已经发表了相当数量的论文并开发出了一系列的软件支持工具。进行软件测试领域知识管理的研究实际上是一个知识管理思想在软件测试领域的 IT 实现问题,也就是开发出一个软件平台来支持软件测试中的知识管理活动。通过知识管理系统平台真正实现有效的知识获取、编码、存储和搜索还有相当长的一段路要走。

2. 存在的主要问题

根据相关人员软件测试项目的工作经验,同时结合知识管理的基本原理分析国际上主流软件测试过程,我们认为目前在软件测试过程中存在五大问题:

(1)软件测试知识重用率低。

(2)软件测试知识传递不畅。

(3)软件测试知识共享环境差。

(4)软件测试知识流失严重。

(5)无法快速实现测试组织中人力资源的优化配置。

由于存在这些问题,造成了软件测试企业的生产效率不高,对市场的整体响应速度慢,应变能力不强。

1.6 本章小结

本章是本书的概论,简要而全面地介绍了软件的含义、软件的发展、软件过程、软件工程、软件工程的开发过程与基本原理、软件工程在软件行业中的作用、软件工程的方法论等

方面的内容。

软件的发展与软件危机相伴而生，软件的发展历程可分为程序设计阶段、程序系统阶段以及软件工程阶段。软件危机是指在计算机软件的开发和维护中所遇到的一系列严重问题，只能缓解软件危机而不可能彻底消除软件危机。

软件工程是工程概念在软件领域里的一个特定应用，涉及软件产品的所有环节，包括过程、方法、工具三个要素，软件工程的开发方法是开发软件的核心步骤。

习题 1

1. 对于硬件是有形的而软件是无形的观点，有人提出了不同的看法。他认为软件也是有形的，例如，软件需要安装才能工作，软件安装之后会占据一定的磁盘空间。对此，你有什么看法？

2. 软件按服务对象的不同可分为通用软件和定制软件。试举例说明这两类软件的区别。

3. 程序系统时代出现的“软件作坊”有什么特点？

4. 某软件公司抢时间为某单位开发了一个人事管理软件。但软件交付用户使用一段时间之后，用户产生了抱怨，原因是单位里某个职工改了名字，但人事管理软件却不允许修改姓名，而只能删除整条记录以后重新输入。试从软件危机角度对这个问题做一些评论。

5. 什么是软件工程？

6. 什么是软件工程方法？简要说明一些主要的软件工程方法。

7. 什么是软件工具？Visual C 属于什么类型的软件工具？

8. 你是如何看待软件工程过程的？软件过程中最基本的活动有哪些？

9. 软件工程管理主要包括哪些方面的内容？

10. B. W. Boehm 提出的软件工程基本原则的作用是什么？

第2章 软件生命周期与开发模型

随着计算机应用范围的不断扩大与深化，软件越来越复杂。软件开发的实践使人们意识到，软件系统的开发与其他工业产品的开发一样，也有必不可少的设计、制作、检验等环节。将软件作为一种产品，它的开发也应当划分阶段，就像工业生产的流水线一样，分阶段制作和检验，最终获得合格的软件产品。软件系统或软件产品也有定义、开发、运行维护，直至被淘汰这样的完整过程，我们把软件将要经历的这个全过程称为软件的生命周期。

为了使软件生命周期中的各项任务能够有序地按照规程进行，需要一定的工作模型对各项任务给以规程约束，这样的工作模型被称为软件过程模型，或软件生命周期模型。它是一个有关项目任务的结构框架，规定了软件生命周期内各项任务的执行步骤与目标。

2.1 软件生命周期模型概论

根据我国国家标准《计算机软件开发规范》(GB 8566-8)，软件生命周期包含：软件定义、软件开发、软件运行维护三个时期，并可以细分为可行性研究、项目计划、需求分析、概要设计、详细设计、编码实现与单元测试、系统集成测试、系统确认验证、系统运行与维护等几个阶段。这个流程就是软件生命周期的基本结构，在实际的软件项目中，根据所开发软件的规模、种类、复杂程度、软件开发组织的经验方法，以及采用的技术手段等因素，可以对各阶段进行必要的合并、分解或补充。

2.1.1 软件定义期

软件定义是软件项目的初期阶段，主要由软件系统需求分析人员和用户交流合作，针对有待开发的软件系统进行分析、规划和规格描述，主要任务是解决“做什么”的问题。确定工程的总目标和可行性；导出实现工程目标应使用的策略及系统必须完成的功能；估计完成工程需要的资源和成本；制定工程项目进度表。此阶段为之后的软件开发工作做准备，通常需要分阶段地进行以下工作。

1. 任务立项与可行性分析

软件项目往往开始于任务立项，必须确定待开发软件系统的总目标，给出它的功能、性能、约束、接口以及可靠性等方面的要求；由软件分析员和用户合作，探讨解决问题的可能方案，针对每个候选方案，从技术、经济、法律和用户操作等方面，研究完成该项软件任务的

可行性，并对可利用的资源（计算机硬件、软件、人力等）、成本、可取得的效益、开发的进度做出估算，制定完成开发任务的实施计划，连同可行性研究报告，提交管理部门审查。并需要针对软件项目的名称、性质、目标、意义和规模等方面撰写相关文档，对准备开发的软件项目进行较为全面的直观描述。

在软件任务立项工作完成之后，需要对软件项目进行可行性分析，针对准备进行开发的软件项目进行可行性风险评估。一般做法是对准备开发的软件系统提出高层模型，并根据高层模型的特征，从技术可行性、经济可行性和操作可行性这三个方面进行分析，并决定项目能否或者是否值得继续进行下去。

2. 软件需求分析

软件需求分析是软件定义期需要达到的目标，要求以用户需求为基本依据，从功能、性能、数据、操作等多个方面，对软件系统给出完整、准确、具体的描述，用于确定软件规格。对用户的要求进行分析，明确目标系统的功能需求和非功能需求，并建立分析模型，从功能、数据、行为等方面描述系统的特性，细化系统的各种需求细节。基于分析结果，编写出软件需求分析文档或系统功能规格文档。

在软件项目开发过程中，需求分析是从软件定义到软件开发的最关键步骤，其结论不仅是今后软件开发的基本依据，同时也是今后用户对软件产品进行验收的基本依据。

3. 制定项目计划

在确定项目可以进行开发以后，接着需要针对项目的开展，从人员、组织、进度、资金、设备等多个方面进行合理的规划，并以“项目开发进度计划”的形式提交书面报告。

2.1.2　软件开发期

在对软件规格完成定义以后，接着可以按照软件需求分析相关文档的要求对软件实施开发，这个时期需要分阶段地完成以下几项工作。

1. 概要设计

概要设计是针对软件系统的结构设计，用于从总体上对软件的构造、接口、全局数据结构和数据环境等给出设计说明，并以“概要设计说明书”的形式提交书面报告，其结果将成为详细设计与系统集成的基本依据。

模块是概要设计时构造软件的基本元素，因此，概要设计中的软件也就主要体现在模块的构成与模块的接口这两个方面。结构化设计中的函数、过程，面向对象设计中的类、对象，它们都是模块。概要设计时并不需要说明模块的内部细节，但是需要进行全面的有关它们构造的定义，包括功能特征、数据特征和接口等。

在进行概要设计时，模块的独立性是一个有关质量的重要技术性指标，可以使用模块的内聚、耦合这两个定性参数对模块独立性进行度量。

2. 详细设计

详细设计以概要设计为依据，用于确定软件结构中每个模块的内部细节，为编写程序提

供最直接的依据。详细设计需要从实现每个模块功能的程序算法和模块内部的局部数据结构等细节内容上给出设计说明，并以“详细设计说明书”的形式提交书面报告。

3. 编码实现和单元测试

编码是对软件的实现，一般由程序员完成，并以获得源程序基本模块为目标。

编码必须按照“详细设计说明书”的要求逐个模块地实现。在基于软件工程的软件开发过程中，编码往往只是一项语言转译工作，即把详细设计中的算法描述语言转译成某种适当的高级程序设计语言或汇编语言。

为了方便程序调试，针对基本模块的单元测试也往往和编码结合在一起进行。单元测试也以“详细设计说明书”为依据，用于检验每个基本模块在功能、算法与数据结构上是否符合设计要求。

4. 系统集成测试

所谓系统集成也就是根据概要设计中的软件结构，把经过测试的模块，按照某种选定的集成策略，例如渐增集成策略，将系统组装起来。在组装过程中，需要对整个系统进行集成测试，以确保系统在技术上符合设计要求，在应用上满足需求规格要求。

5. 软件发布与实施

在完成对系统的集成与测试之后，接着就要对系统进行验收、发布与实施。系统验收与发布需要以用户为主体，以需求规格说明书中对软件的定义为依据，由此对软件的各项规格进行逐项地确认，以确保已经完成的软件系统与需求规格的一致性。为了方便用户在系统确认期间能够积极参入，也为了系统在以后的运行过程中能够被用户正确使用，这个时期往往还需要以一定的方式对用户进行必要的培训。

在完成对软件的验收之后，软件系统可以交付用户使用，并需要以“项目开发总结报告”的书面形式对项目进行总结。

2.1.3 软件运行与维护期

软件系统的运行是一个比较长久的过程，跟软件开发机构有关的主要任务是对系统进行经常性的有效维护。

软件的维护过程，也就是修正软件错误、完善软件功能，由此使软件不断进化升级的过程，以使系统更加持久地满足用户的需要。因此，对软件的维护也可以看成是对软件的再一次开发。在这个时期，对软件的维护主要涉及三个方面的任务，即改正性维护、适应性维护和完善性维护。

2.2 瀑布模型

瀑布模型(Waterfall Model)也称线性顺序模型，是 Winston Royce 于 1970 年提出的。这种过程模型的各个阶段的工作顺序开展，就像阶梯式的瀑布，水由上向下一个阶梯一个阶

梯地倾泻而下，最后进入一个平静的大湖，也就是实际中的软件企业的软件产品库。传统的瀑布模型如图 2-1 所示。

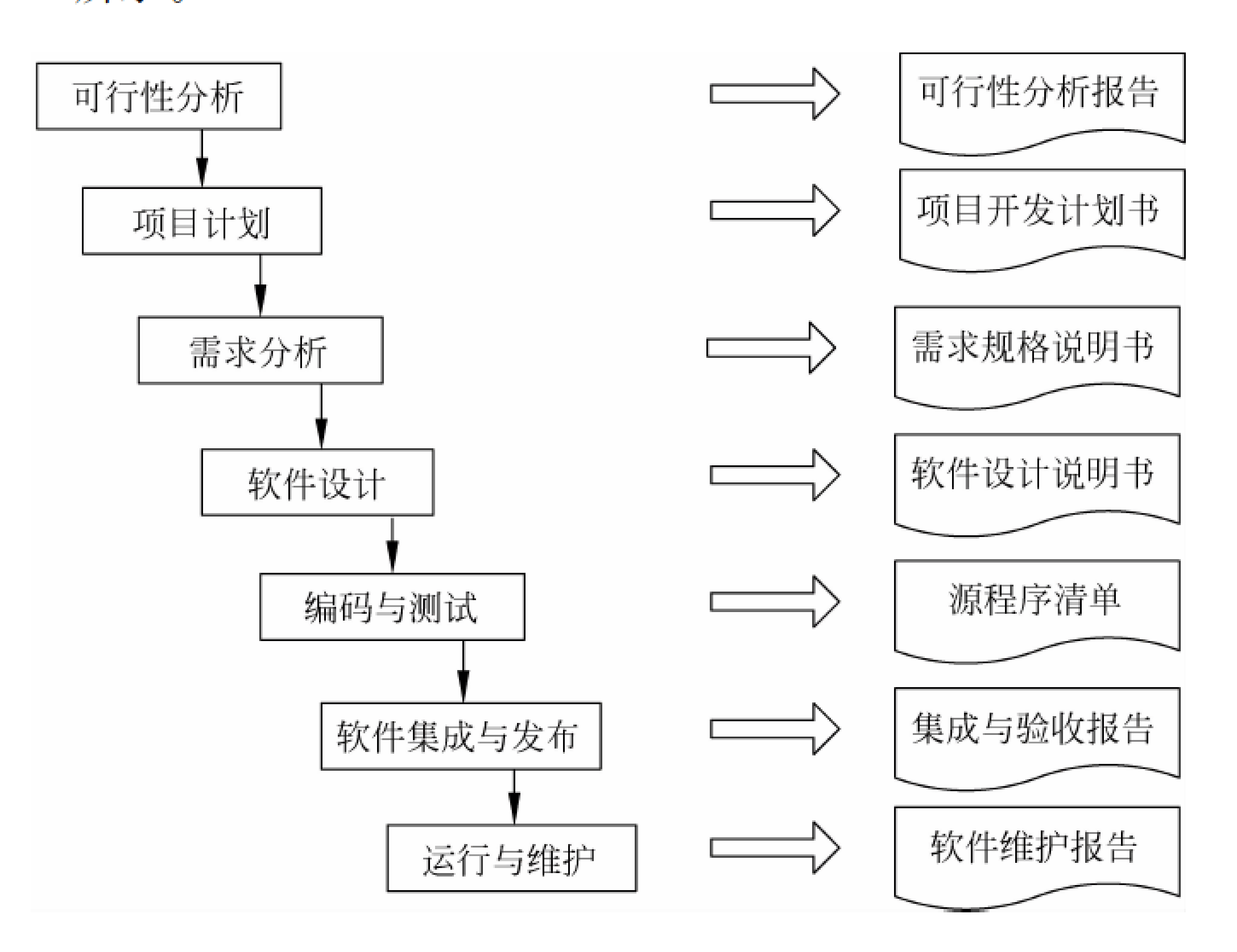


图 2-1　传统的瀑布模型图

在瀑布模型中，软件生命周期的过程是由可行性分析、项目计划、需求分析、软件设计、编码与测试、软件集成与发布、运行与维护等阶段组成。瀑布模型中的“瀑布”是对这个模型的形象表达，把软件生存过程比喻成瀑布中的流水，软件生存过程在这些台阶中从上往下地奔流。瀑布模型规定了各项开发工程活动，自顶向下、逐层细化、相互衔接、逐级下落，当发现某一阶段的上一级活动存在缺陷时，可以追溯，进行消除或改进，但同时也会付出代价，需要消耗更多的资源。瀑布模型中的逐层细化，其含义则是对软件问题的不断分解而使问题不断具体化、细节化，以方便问题的解决。

2.2.1　瀑布模型的特点

1. 线性化模型结构

瀑布模型所考虑的软件项目是一种稳定的线性过程，最大的特点就是简单直观。项目被划分为从上至下按顺序进行的几个阶段，阶段之间有固定的衔接次序，并且前一阶段输出的成果被作为后一阶段的输入条件。

2. 各阶段具有里程碑特征

瀑布模型中的阶段只能逐级到达，不能跨越。每个阶段都有明确的任务，都需要产生出确定的成果，其标志主要有：需求规格说明书、软件设计说明书、开发管理计划书、程序代码清单、集成与测试报告、维护报告等。

3. 基于文档的驱动

文档在瀑布模型中是每个阶段的成果的直观体现，也就成为了各个阶段的里程碑标志。

由于后一阶段工作的顺利开展是建立在前一阶段所产生的文档基础上，因此，文档也就成为了推动下一阶段工作开展的保障和动力。

4. 严格的阶段评审机制

在完成某个阶段的工作任务并准备进入到下一个阶段之前，需要对这个阶段的项目文档进行严格地评审，直到确认以后才能启动下一阶段的工作。为了保证质量，瀑布模型除了安排单独的软件测试阶段之外，还在开发过程中的各个阶段结束以及在各个阶段的多个活动中插入若干技术复审或验证环节，以便尽早发现问题，消除隐患。

2.2.2 瀑布模型的优缺点

1. 模型的优点

瀑布模型的优点主要有：开发阶段界定清晰，有利于项目的评审、验证、跟踪、管理和控制，这也决定了它是一种软件工程应用中的主流开发模型。

瀑布模型每个开发阶段都有指定的起点和终点，通过里程碑的标志可以被客户和开发者识别，在编写程序代码之前充分强调和重视需求和设计，避免了浪费时间，并且保证了客户的实际需求得以满足。

在瀑布模型中因果关系紧密相连，前一个阶段工作的结果是后一个阶段工作输入。或者说，每一个阶段都是建立在前一个阶段正确结果之上的，前一个阶段的错误和漏洞会被隐蔽地带到后一个阶段。这种错误有时甚至可能是灾难性的。因此，每一个阶段工作完成后，都要进行审查和确认，这是非常重要的，有利于对开发人员的组织管理，有利于对软件开发方法和工具的研究。

2. 模型的缺点

瀑布模型是一种线性的过程，适用于在开发的早期阶段软件需求已经完全确定的情况。但在实际开发中，这种要求过于理想化，难以适应现代软件开发的实际情况，其暴露出的主要问题如下所示：

（1）工作流程只能一个一个台阶地往下流动，不可能倒着向上流动，这是瀑布模型的致命缺点。

（2）通常会导致项目后期，如在项目实施阶段，出现“问题堆积”，在之前的各个工作阶段中隐藏下来的问题，会在后期逐渐暴露出来，甚至会发散扩大。

（3）由于开发模型是线性的，用户只有等到整个过程的末期才能见到开发成果，中间提出的变更要求很难得到响应，因此增加了开发的风险。

（4）用户和软件项目负责人要等待相当长的时间才能得到初始版本。这时系统如果改变需求，将会带来巨大的资源损失。

2.2.3 瀑布模型的适用范围

瀑布模型并不是适合所有的软件项目，瀑布模型有以下一些适用范围：

- 在软件项目开发期间需求没有变化或很少变化；

- 开发者对应用领域很熟悉(例如,开发任务是扩展一个已存在的系统);
- 低风险项目(例如,开发者对目标和开发环境非常熟悉);
- 用户使用软件的环境很稳定;
- 用户提出需求以后,很少参与开发工作;
- 软件系统编程要求使用面向过程的程序设计语言。

尽管瀑布模型的应用条件比较严格苛刻,但是软件企业在开发软件产品时,往往还是采用瀑布模型,系统软件和工具软件的开发工作也常常采用瀑布模型。

瀑布模型较难适应用户的需求变更,开发速度慢。但是,瀑布模型提供了一套工程化的里程碑管理模式,能够有效保证软件质量,并使得软件容易维护。

2.3　增量模型

增量模型(Incremental Model)是遵循递增方式来进行软件开发的。它是把瀑布模型的顺序特征和原型模型的迭代特征,相结合的一种软件构件化的递增式模型。把软件描述、设计、实现活动分解成一系列相互联系的增量构件的迭代开发过程,将软件产品作为一组增量构件或模块,每次需求分析、设计、实现、测试和交付发布一个构件,直到所有构件全部实现时终止。

2.3.1　增量模型的特点

增量模型是瀑布模型和原型进化模型的综合,采用随着日程时间的进展而交错的线性序列,每一个线性序列产生软件的一个可发布的“增量”。增量模型在整体上按照瀑布模型的流程进行项目的开发和管理,但在软件的实际开发工作中,是将软件系统按功能分解为许多增量构件,并以构件为单位逐个地开发与交付,直到全部增量构件开发完毕,并都被集成到系统之中交付用户使用。

当使用增量模型开发软件时,第一个增量往往是核心的产品,即实现了基本的需求,但被很多补充的特征还没有被发布。客户对每一个增量的使用和评估都作为下一个增量发布的新特征和功能,这个过程在每一个增量发布后不断重复,直到产生了最终的完善产品。增量模型强调每一个增量均发布一个可操作的产品。增量模型的工作流程如图 2-2 所示。

增量模型的工作流程可分为以下三个阶段:

(1) 在软件开发的前期阶段,需要对整个系统进行需求分析和概要设计,分析确定系统的增量构件组成的需求框架,之后以需求框架中构件的组成及关系为依据,完成对软件系统的体系结构设计。

(2) 在软件开发的增量构件设计阶段,需要对构件进行需求分析,然后进行设计、编码实现、测试和正确性验证工作。

(3) 完成某个增量构件的开发后,需要将该构件集成到软件系统中,并对集成新增量构件的系统重新进行测试和验证工作,之后再继续下一个增量构件的开发。

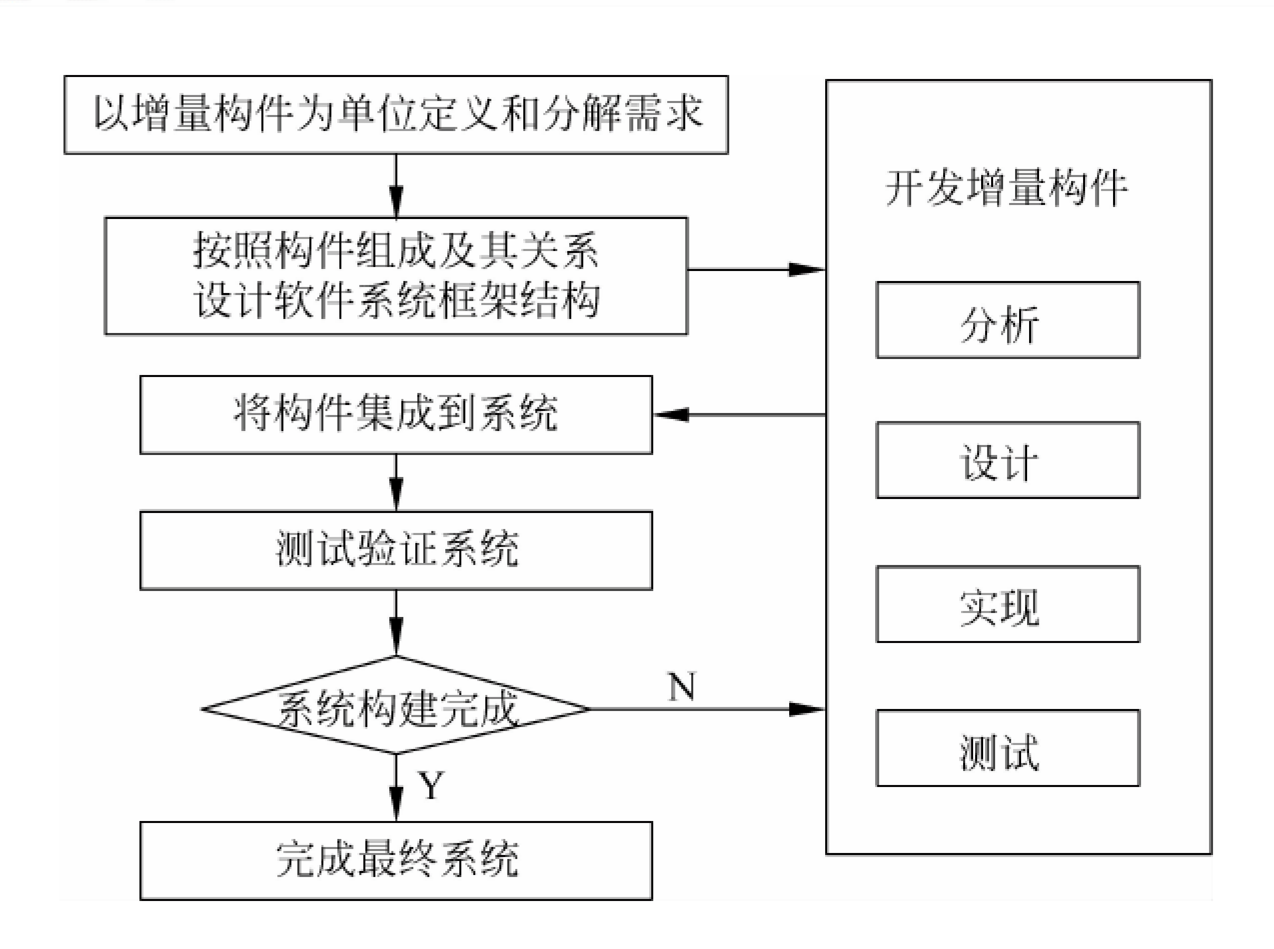


图 2-2　增量模型的工作流程

2.3.2　增量模型的优缺点

1. 模型的优点

在增量模型中人员可以灵活分配，刚开始不用投入大量人力资源，当核心模块产品很受欢迎时，可增加人力实现下一个增量。

增量模型将整个产品分解成若干个构件进行逐步交付，因此软件开发可以较好地适应需求的变化，用户可以不断地看到开发软件所对应的可运行的中间版本，从而降低了开发风险。

2. 模型的缺点

由于各个构件是逐渐并入已有的软件体系结构中的，所以加入构件不能破坏已构造好的系统部分，这需要软件具备开放式的体系结构。

在开发过程中，需求的变化是不可避免的。增量模型的灵活性可以使其适应这种变化的能力大大优于瀑布模型，但是也很容易转化为边做边改的方式，从而使软件过程的控制失去整体性。

2.4　原型模型

许多软件公司在生产软件产品或实施软件项目开发时，经常采用一种“原型法”，它就来源于原型模型。

由于瀑布模型在软件开发中的缺点，人们在借鉴建筑师、工程师建造原型的经验基础上，提出了原型模型。原型模型（Prototype Model）又称为快速原型，意思是软件开发人员根据客户提出的初步需求，快速开发一个原型，向客户展示待开发软件系统的全部或部分功能或性能，并在征求客户对原型意见的过程中，进一步修改、完善，之后让客户试用，反复循环这个过程几次或多次，直到客户确认软件系统为止。

2.4.1　快速原型方法

由于原型模型的开发速度较快，有时也被称为快速原型法。快速原型方法是原型模型在软件分析、设计阶段的应用，用来解决用户在需求上对软件系统的模糊认识，或用来试探某种设计是否能够获得预期成果。

快速原型方法具有以下特点：

(1) 快速原型是用来获取用户需求的，或是用来试探设计是否有效的。一旦需求或设计确定下来，原型就将被抛弃。因此，快速原型要求快速构建、容易修改，以节约原型创建成本、加快开发速度。在开发工具和开发环境迅速发展的现在，快速原型往往采用一些高效生成工具创建，例如 ER win、Power Designer 等数据库分析工具。Microsoft Visual Studio、Delphi 等基于组件的可视化开发工具，也被应用于原型创建之中，而且还可用于原型进化。

(2) 快速原型是暂时使用的，因此并不要求完整。它往往针对某个局部问题建立专门的原型，如界面原型、工作流原型、查询原型等。

(3) 快速原型不能贯穿软件的整个生命周期，它需要和其他的过程模型相结合才能产生作用。例如，在瀑布模型中应用快速原型，以解决瀑布模型在需求分析时期存在的不足。

2.4.2　原型进化模型

原型进化对开发过程的考虑是，针对有待开发的软件系统，先开发一个原型系统给用户使用，然后根据用户使用情况的意见反馈，对原型系统不断修改，使它逐步接近并最终到达开发目标。快速原型在完成需求定义后将被抛弃，与快速原型不同的是原型进化所要创建的原型则是一个今后将要投入应用的系统，只是所创建的原型系统在功能、性能等方面还有许多不足，还没有达到最终开发目标，需要不断改进。原型进化模型工作流程如图 2-3 所示。

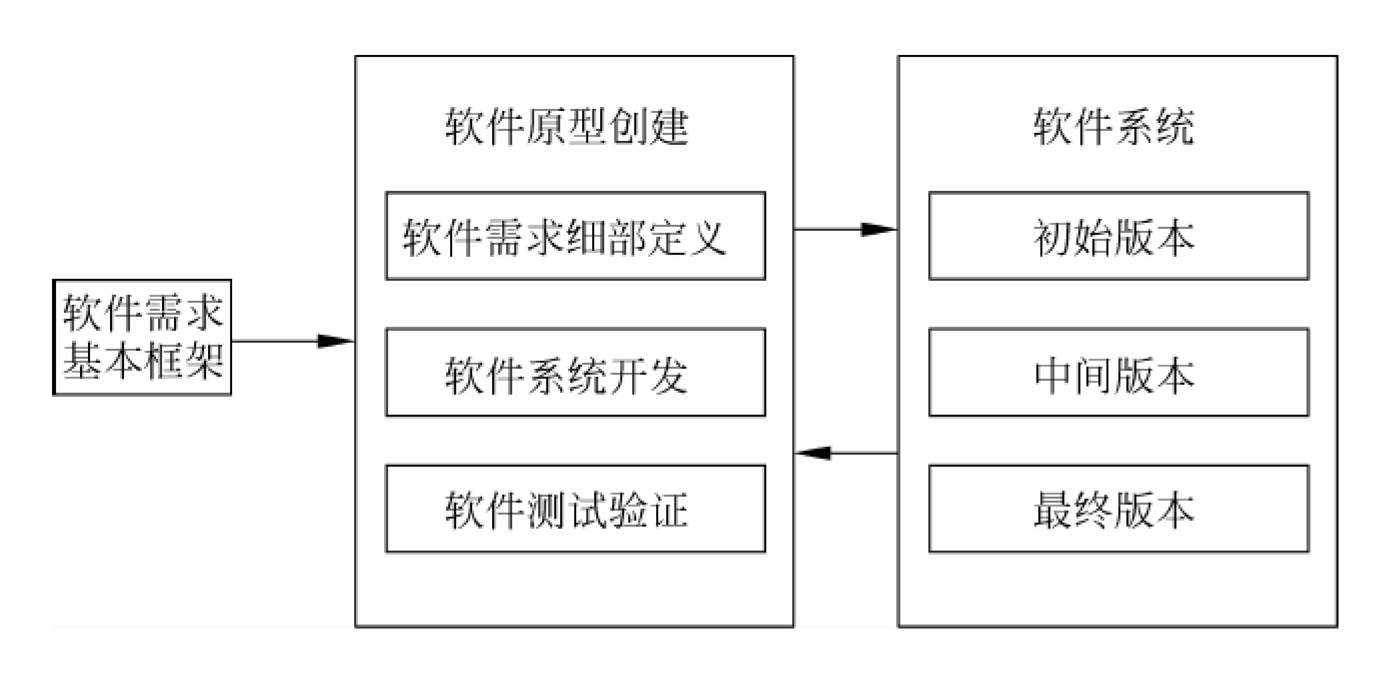


图 2-3　原型进化模型工作流程

原型进化模型有两个特点：

(1) 原型进化模型将详细定义软件需求的详细定义、产品开发和测试验证放在同一个工作流程中交替或并行运作。在获得了软件需求基本框架以后，也就是软件的基本功能被确定以后，就可以直接进入软件的开发。

(2) 原型进化模型是通过不断发布新的软件版本而使软件逐步完善的,这种开发模式特别适合于那些对用户急需的软件系统的开发。它能够快速地向用户交付可以投入实际运行的软件成果,并能够很好地适应软件用户对需求规格的变更。

原型进化模型能够适应软件需求的中途变更,但在实际应用的时候,还要重视两个问题:

(1) 原型进化模型虽然可以加快开发进程,但不能像瀑布模型那样提供明确的里程碑式的管理,随着开发过程中软件版本的快速更新,项目管理、软件配置管理会变得越来越复杂,管理者难以把握开发进度。因此,对于大型软件项目,原型进化模型缺乏有效的规范化管理。

(2) 开发过程中软件版本的快速更新,还可能损伤软件的内部结构,使其缺乏整体性和稳定性。此外,用于反映软件版本变更的文档也有可能跟不上软件的变更速度。这些问题必将影响到后期软件的维护工作。

2.5 迭代模型

在软件项目开发的实际工作中,针对瀑布模型的缺陷,人们提出了迭代模型(Iterative Model)。在多种迭代模型中,美国的 I. Jacobson、G. Booch 和 J. Rumbaugh 三位软件专家提出的 RUP(Rational Unified Process)模型最为成功。他们在 1995 年提出了统一建模语言(Unified Modeling Language,UML)的雏形,之后该语言在 Rational Rose 开发工具中得到了初步实现,之后在迭代模型的启发下,1997 年提出了 USDP(the United Software Development Process,统一软件开发过程),之后 USDP 更名为 RUP。

RUP 将迭代被定义为:迭代包括产生产品发布(稳定、可执行的产品版本)的全部开发活动和要使用该发布所必需的所有其他外围元素。所以,在某种程度上,开发迭代是一次完整地经过所有工作流程的过程,应该包括需求工作流程、分析设计工作流程、实施工作流程和测试工作流程。从这个意义上讲,原型不断完善,增量不断创建,都是迭代的过程,所以原型法和增量模型都可以看做是局部迭代模型。并且 RUP 推出的一种"逐步求精"的面向对象的软件开发过程,是迄今为止最完善、可实现商品化的开发过程模型。

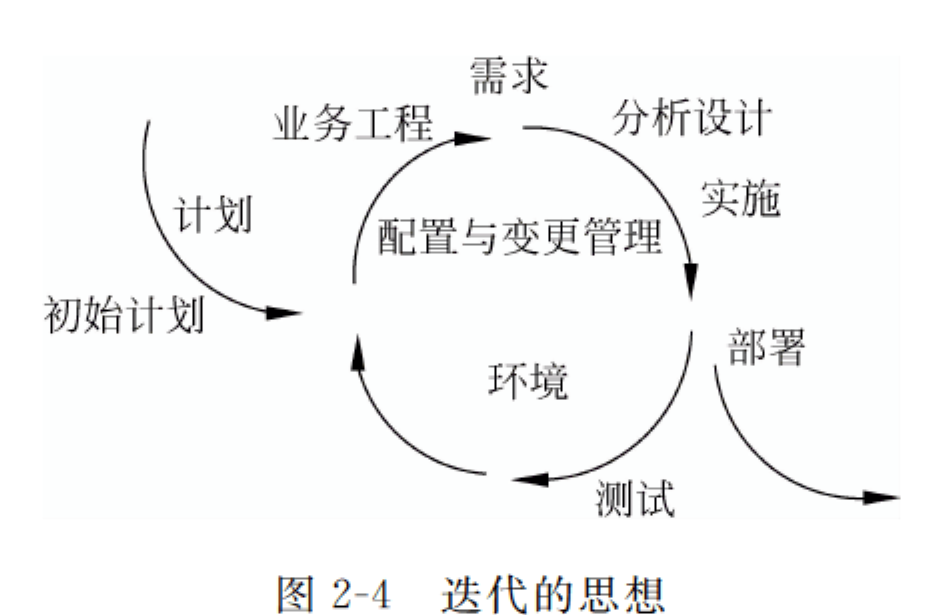


图 2-4 迭代的思想

实质上,它类似小型的瀑布式项目。RUP 认为,所有的阶段(需求及其他)都可以细分为迭代。每一次的迭代都会产生一个可以发布的产品,这个产品是最终产品的一个子集。迭代的思想如图 2-4 所示。

2.5.1 迭代模型的阶段及核心流程

1. 模型的主要阶段

(1) 初始阶段。该阶段的主要工作是确定系统的业务用例(Use Case)和定义项目的范围。需要标识系统要交互的外部实体,定义高层次的交互规律,定义所有用例并对其中重要

的用例进行描述和实现。还需要标识业务用例，包括成功的评估、风险确认、资源需求和阶段实施计划。

(2) 精化阶段。该阶段的主要工作是分析问题域、细化产品定义、定义系统的构架并建立基础，为构建阶段的设计和实施提供一个稳定的基础。为了验证系统架构，可能还需要实现基本的原型。

(3) 构建阶段。该阶段主要工作是反复开发，以完善软件系统，达到用户的要求。包括用例的描述、完成设计、编码实现和对软件进行测试等工作。

(4) 产品化阶段。该阶段的主要工作是将软件产品交付给用户，包括安装、培训、交付、维护等工作。

2. 模型的核心流程

迭代模型事先要有一个初始业务模型，以便于进行迭代流程，模型的核心流程如下。

(1) 业务建模。目的是对用户的需求及业务流程的重新规划与合理改进，也是对业务流程的优化与抽象，使开发的软件系统能反映优化的、合理的业务流程。

(2) 获取需求。与客户在系统中的工作内容、系统的业务功能方面达成一致，使开发人员能够更清楚地了解需求，定义系统的用户界面、用户的需求和目标。

(3) 分析设计。将需求转换为待开发系统的设计，逐步开发系统的架构，保证设计适合实施环境，保证系统功能和性能满足需求。

(4) 实施。对照实施子系统的分层结构定义代码结构，以构件方式实施类和对象，对已开发的构件按单元进行测试，将已经完成的系统构件集成到可执行系统中。

(5) 测试。将构件部署到应用系统的真实环境中，使用户可以将软件系统应用于工作流程。

(6) 配置与变更管理。目的是始终保持工作产品的完整性和一致性。

(7) 项目管理。为软件密集型项目的管理提供框架，为项目计划、人员配备、执行和评审提供实用准则，为风险管理提供框架。

(8) 环境。为软件开发机构提供软件开发环境，包括流程环境和工具环境。

2.5.2 迭代模型的优缺点

迭代模型的优点：在开发的早期或中期，用户需求可以变化。在迭代之初，不要求有一个近似的产品原型。模型的适用范围很广，几乎适用于所有软件项目的开发工作。

迭代模型的缺点：迭代模型采取循环工作方式，每次循环都使工作产品更靠近目标产品，这要求项目组成员具备很高的技术水平，掌握先进的开发工具。否则会存在较大的技术和技能风险。

统一软件开发过程(RUP)的内容非常丰富，定义了初始、精化、构建、产品化四个阶段和业务建模、获取需求、分析设计、实施、测试、部署九个流程，提供了大量的文档模板，但极易让人误解是重型的过程，实施推广有一定难度。

2.6 螺旋模型

软件开发过程中存在许多方面的风险。例如，遇到了很难克服的技术难题、开发成本超出了前期预算、软件产品不能按期交付、用户对所交付的软件不满意等。软件风险是任何软件项目中都普遍存在的实际问题，而且项目越大，软件越复杂，风险也就越大。软件风险会损害软件的开发过程，会影响软件产品质量。在软件开发过程中需要及时地识别风险、有效地分析风险，并能够采取适当措施消除或减少风险的危害。

螺旋模型是一种引入了风险分析与规避机制的过程模型，是瀑布模型、快速原型方法和风险分析方法的有机结合。螺旋模型的基本方法是，在各个阶段创建原型进行项目试验，以降低各个阶段可能遇到的项目风险。例如，为了降低用户对软件界面不满意的风险，可以在需求分析阶段建立"界面原型"；为了降低软件不能按设计要求实现的风险，可以在设计阶段针对所采用的技术建立"仿真试探原型"。

螺旋模型的工作流程如图 2-5 所示。它用螺旋线表示软件项目的进行情况，其中，螺旋线中的每个回路表示软件过程的一个阶段。因此，最里面的回路与项目可行性有关，接下来的一个回路与软件需求定义有关，而再下一个回路则与软件系统设计有关，以此类推。

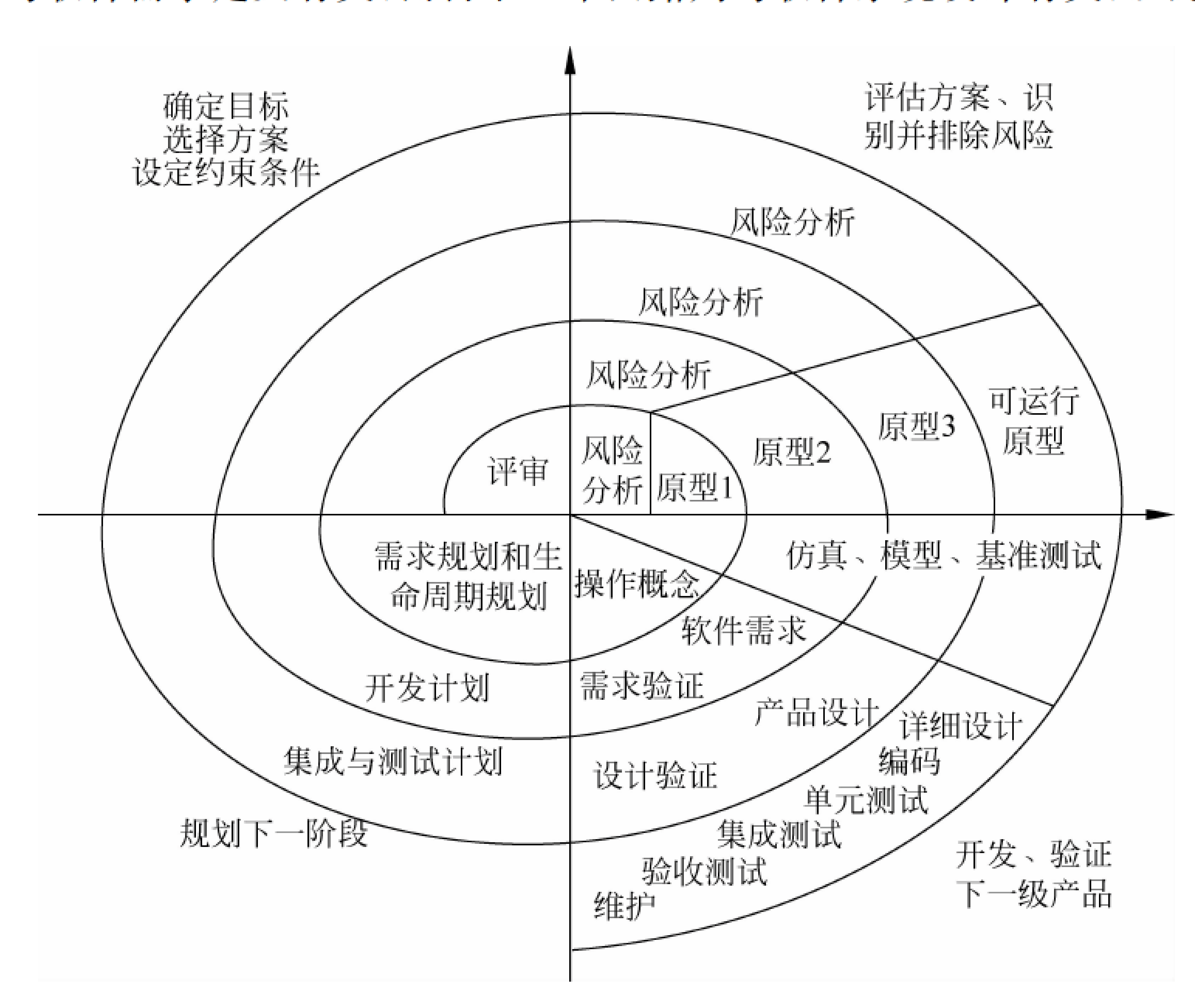


图 2-5 螺旋模型的工作流程

螺旋线中的每个回路都被分成为以下四个部分。

(1) 制定计划：确定软件目标，选择实施方案，明确项目开发的限制条件。

(2) 风险分析：对风险进行详细的评估分析，并确定适当的风险规避措施。

(3) 实施工程：实施软件开发和验证工作。

(4) 客户评价：评价开发工作，提出修正意见，制定下一步计划。

2.6.1 螺旋模型的特点

模型具有以下特点：

(1) 将开发过程组织成一个逐步细化的螺旋周期，每经过一个周期，系统就得到进一步的细化完善。

(2) 整个模型紧密围绕开发中的风险分析，推动软件产品向深层扩展和求精。

(3) 强调持续的判断、确定和修改用户的任务目标，并按成本、效益来分析软件产品对任务目标的贡献。

2.6.2 螺旋模型的优缺点

1. 螺旋模型的优点

(1) 与瀑布模型相比，螺旋模型支持用户需求的动态变化，为用户参与软件开发的决策提供了方便，有助于提高软件的适应能力。

(2) 螺旋模型对可选方案和约束条件的强调，有利于对已有软件的重用，有助于把软件质量作为软件开发的一个重要目标。

(3) 减少了过多测试或测试不足的情况所带来的风险。

(4) 在螺旋模型中，维护只是模型的一个周期，在维护和开发之间没有本质区别，软件维护得到了重视。

2. 螺旋模型的缺点

(1) 螺旋模型强调风险分析，但要求许多客户接受和相信这种分析并做出相应的反应是不容易的，因此这种模型往往适应于内部的大规模软件开发。

(2) 如果执行风险分析会大大影响项目的利润，那么进行风险分析便毫无意义，因此螺旋模型只适用于大规模软件项目。

(3) 过多的迭代次数会增加开发成本，延迟软件交付时间。

2.7 喷泉模型

喷泉模型(Fountain Model)是一种以用户需求为动力，以对象为驱动的模型，主要用于描述面向对象的软件开发过程。“喷泉”一词用于形象地表达面向对象软件开发过程中的迭代和无缝过渡。

喷泉模型认为，软件开发过程自下而上周期的各阶段是相互重叠和多次反复进行的，就像喷泉中的水喷上去又落下来，所以叫做喷泉模型。开发的各个阶段没有特定的次序要求，可以交互进行。还可以随时补充其他任何开发阶段中的遗漏。采用喷泉模型的软件过程如图 2-6 所示。

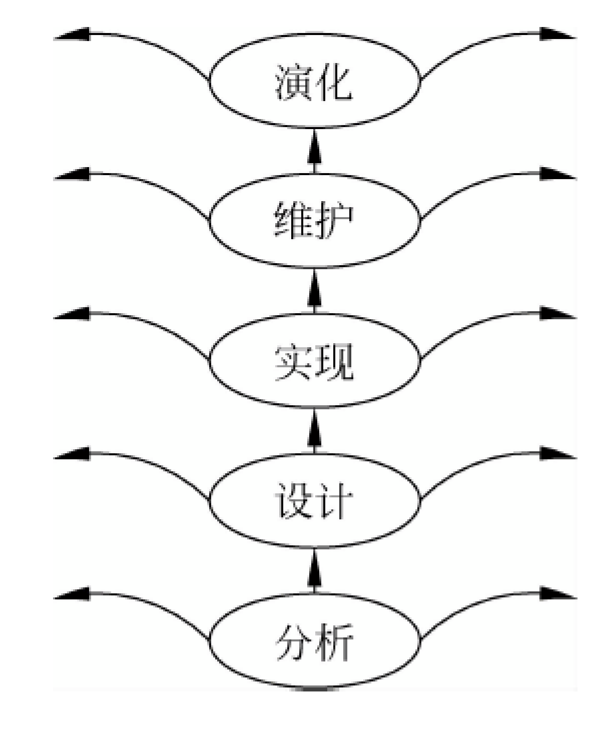


图 2-6 喷泉模型的软件过程

2.7.1 喷泉模型的特点

喷泉模型是一种以用户驱动的模型，主要用于描述面向对象的软件开发过程。由于各阶段的活动之间无明界线，所以喷泉模型也被称为“喷泉无间隙性模型”。

喷泉模型的过程方法所考虑的是，基于面向对象方法对软件进行分析、设计和实现按照迭代的方式交替进行，并通过进化的方式，使软件分阶段逐渐完整、逐步求精。例如，第一阶段软件开发的目标可以是实现软件的基本功能；第二阶段可以是在第一阶段建立的软件的基础上，对软件进行进一步完善，并实现软件的主要功能；第三阶段则是在第二阶段的基础上，对软件进行更加完整地开发，并以实现软件全部功能作为创建目标。

应该说，喷泉模型能够较有效地平衡软件系统的近期需求与远期规划，因此能够较好地满足用户在软件应用上的发展需要。

2.7.2 喷泉模型的优缺点

1. 喷泉模型的优点

喷泉模型不像瀑布模型那样，需要分析活动结束后才开始设计活动，在设计活动结束后才开始编码活动。该模型的各个阶段没有明显的界限，开发人员可以同步进行开发。其优点是可以提高软件项目开发效率，节省开发时间，适应于面向对象的软件开发过程。

2. 喷泉模型的缺点

由于喷泉模型在各个开发阶段是重叠的，因此在开发过程中需要大量的开发人员，因此不利于项目的管理。此外这种模型要求严格管理文档，使得审核的难度加大，尤其是面对可能随时加入各种信息、需求与资料的情况。

2.8 XP 模型

XP 模型(eXtreme Programming Model，极限编程模型)本来是敏捷企业文化现象，但是现在不少人将它当做一种软件开发模型。

敏捷方法是近几年兴起的一种轻量级的开发方法。它强调适应性而非预测性，强调以人为中心，而不以流程为中心，以及对变化的适应和对人性的关注。其特点是轻载、基于时间、适量(Just Enough)、并行并基于构件的软件过程。在所有的敏捷方法中，XP 方法是最引人注目的一种轻量级开发方法。

对传统软件开发模型进行重新审视发现，它们太正规、太呆板、太浪费资源，从而提出了轻量级省时省力的 XP 模型。它由一组简单规则(需求、实现、重构、测试、发布)组成，既能保持开发人员的创造性，又能保持对需求变动的适应性，即使在开发的后期，也可以适应用户需求的变更。XP 模型的迭代开发过程，如图 2-7 所示。

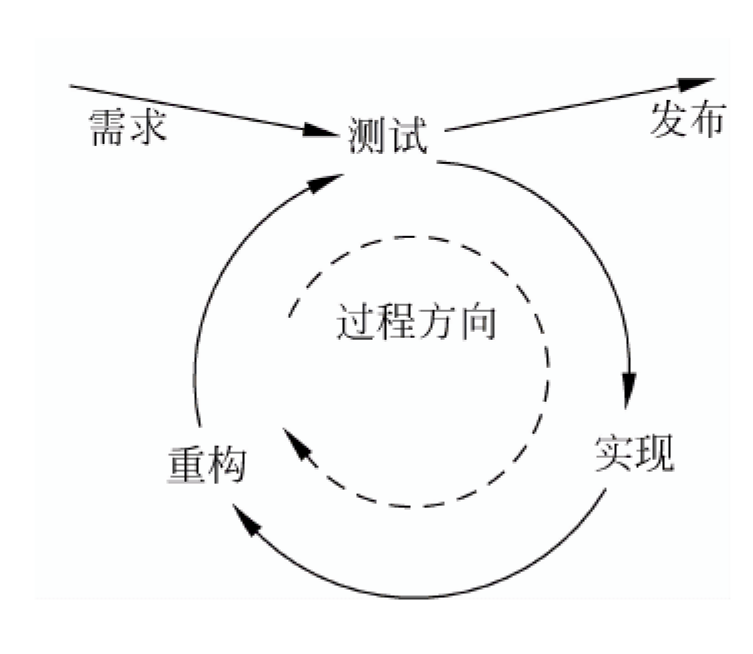


图 2-7 XP 模型的迭代开发过程

2.8.1 XP 模型的特点

在需求、实现、重构、测试、发布的迭代过程中，XP 模型有四条核心原则：交流(Communication)、简单(Simplicity)、反馈(Feedback)和进取(Aggressiveness)。XP 开发小组包括开发人员、管理人员和客户。XP 模型强调小组内成员之间要经常进行交流，组队编程，在尽量保证质量的前提下力求过程和代码的简单化。来自客户、开发人员和软件用户的具体反馈意见，可以提供更多的机会用于调整设计，保证正确的开发方向。进取则包含在上述三个原则中。在 XP 模型中采取讲"用户场景故事"的方法，来代替传统模型中的需求分析。这个过程由用户自己讲，不用考虑技术细节，只是描绘用户场景。

2.8.2 XP 模型的优缺点

XP 模型具有以下优点：

(1) 采用简单计划策略，不需要长期计划和复杂模型，开发周期短。

(2) 在全过程采用迭代增量开发、反馈修正和反复测试的方法，软件质量有保证。

(3) 能够适应用户经常变化的需求，提供令用户满意的高质量软件。

XP 模型的缺点：XP 模型作为一种新的模型，在实际应用中还存在一些问题，引起了一些争议。它一般适用于开发小型项目，并且该模型与 ISO 9001、CMMI 的规范也存在冲突。

2.9 各种模型之间的关系

2.9.1 瀑布模型与迭代模型

瀑布模型与迭代模型是两种最基本的开发模型，两者之间关联非常紧密。

在宏观上，迭代模型是动态模型，瀑布模型是静态模型。迭代模型一方面需要经过多次反复的迭代，才能形成最终产品。另一方面，它的每一次迭代，实质上都是执行一次瀑布模型，都要经历初始、精化、构造、交付四个阶段，也就是瀑布模型的过程。

在微观上，迭代模型与瀑布模型都是动态模型。迭代模型与瀑布模型在每一个开发阶段内部，都有一个小小的迭代过程，只有经历这一迭代过程，该阶段的开发工作才能做细、做好。

瀑布模型与迭代模型之间的关系，通俗地讲就是：你中有我、我中有你。反映了人们对客观事物的认识论：认识与掌握某一客观事物，必须经历由宏观到微观的多次反复的过程。只有从宏观上反复迭代几次，才能看清事物全貌，掌握事物的宏观规律。只有从微观上反复迭代几次，才能彻底了解每个细节，掌握事物的微观发展规律。

2.9.2 瀑布模型与增量模型

瀑布模型与增量模型之间也存在着一定的关系。增量模型首先开发核心模块，之后再开发其他模块，这样一个一个地进行开发，直至所有模块开发完毕。然而，在开发每一模块时，开发者一般都采用瀑布模型，从需求、设计、编码、测试一个阶段接着一个阶段地实现。

所以增量模型中有瀑布模型思想，增量模型也体现了迭代思想，每增加一个模块就进行一次迭代，执行一次瀑布模型。

2.9.3 瀑布模型与原型模型

瀑布模型与原型模型之间也有一定的关系。原型模型开始有一个原型，在此基础上以后的每一次迭代，都可能是一次瀑布模型的开发方式。原型模型中不但体现了迭代模型思想，也体现了瀑布模型思想。

2.9.4 瀑布模型与螺旋模型

螺旋模型是瀑布模型和快速原型模型的结合，快速原型模型是原型模型的简化，原型模型又是迭代模型和瀑布模型的组合，这些模块之间是相互依存、彼此相关的。螺旋模型每一次顺时针方向旋转，相当于顺时针方向迭代一次，都是走完一次瀑布模型，这就是它们之间的关系。

2.9.5 XP模型与迭代模型

XP模型是一个自由式迭代模型，它比传统的迭代模型更简单、自由，甚至毫无约束。

2.9.6 生命周期模型之间的关系总结

软件工程虽然来源于工业生产过程、建筑工程、计算机硬件工程，但是又与这些工程不完全相同。软件开发过程不可能完全按照事先设计好的软件蓝图进行，而是一边开发、一边修改软件设计蓝图、一边再按照修改的软件蓝图继续开发，按照这样的顺序循环多次，循环中又包含各种生命周期及开发模型，最后才能生产出成功的产品。

2.10 本章小结

除了本章所述的七种生命周期模型外，另外还有演化模型(Evolutionary Model)和渐增模型(Incremental Model)。软件生命周期虽然多种多样，但在本质上可以归纳为两大类型，瀑布类型和迭代类型。属于瀑布类型的有瀑布模型、增量模型和喷泉模型等。属于迭代类型的有迭代模型、原型模型、螺旋模型、渐增模型、演化模型、XP模型等。

本章介绍了七种软件生命周期模型：瀑布模型、增量模型、迭代模型、原型模型、螺旋模型、喷泉模型和XP模型。其中最常用的是瀑布模型和原型模型，其次是增量模型，最难掌握的是迭代模型。

习题2

1. 什么是软件生命周期？根据国家标准《计算机软件开发规范》，软件生命周期主要包括哪些阶段？

2. 瀑布模型有哪些特点？对于里程碑，你有什么认识？一般认为，瀑布模型不太适用于用户需求经常变更的软件项目，其原因是什么？

3. 试说明快速原型的作用。

4. 原型进化模型是一种与瀑布模型有着显著差别的软件过程模型。与瀑布模型相比，其优点是什么？一般认为，原型进化模型不能适应较大型软件项目的开发，其原因是什么？

5. 增量模型是一种结合了瀑布模型与原型进化模型共同优点的过程模型，其特点是什么？在使用增量模型进行软件开发时需要注意的问题是什么？

6. 试说明螺旋模型的特点。一般认为，只有大型项目才有采用螺旋模型的必要，其原因是什么？

7. 喷泉模型是专门针对面向对象软件开发方法而提出的，其特点是什么？

8. 为什么说组件复用模型是一种有利于软件按工业流程生产的过程模型？

9. 某大型企业计划开发一个“综合信息管理系统”，涉及销售、供应、财务、生产、人力资源等多个部门的信息管理。该企业的想法是按部门优先级别逐个实现，边应用边开发。对此，需要一种比较合适的过程模型。请对这个过程模型做出符合应用需要的选择，并说明选择理由。

第3章 软件立项与合同

在软件项目的开始阶段，必须要做好项目的立项工作，这是项目后续工作的基础。因此，在项目立项时，首先要根据用户的需求、市场的情况以及相关的政策等诸多因素选择合适的项目；其次要对项目的技术性、项目的投资效益和可能的风险进行可行性分析，根据分析的结论制定项目章程，完成立项工作；然后要确定项目的范围和项目的产品以及活动，项目组织要与用户签订合同；最后制订项目的管理计划，开始实施项目开发设计过程。

3.1 软件立项方法与文档

项目作为国民经济和社会发展的基本要素，对于任何一个国家和任何一个企业的发展都起着非常重要的作用。项目的运作离不开科学的管理，而项目的管理水平直接影响着项目的成败。为了掌握软件项目立项的方法，首先需要了解项目和软件项目的基本概念和基本特征。

3.1.1 项目的基本概念

在21世纪的人类社会中，项目可以说是无处不在。建设一条铁路、一段公路、一座桥梁或者其他建筑物都属于项目；设计一款软件、申报和研究一个课题、撰写一本专著和一篇论文也属于项目；举办一场学术研讨会、一个百周年大典或者组织一次旅游也属于项目。这些活动都是要求在一定的时间和一定的费用内完成，并且具有特定的功能、性能和质量标准，通过一次性努力，满足特定的计划和目标。如果本次努力失败，则这个项目就将以失败而告终。

美国项目管理协会(Project Management Institute，PMI)对项目的定义是：项目是为完成一个独特的产品、服务或者任务而进行的一次性努力。实际上，项目是一个特定的、待完成的有限任务，它是指在一定的时间内，为满足特定目标所做的多项相关工作的总称。项目包含三方面的含义。

(1) 项目是一项有待完成的任务，它有着特定的环境和背景要求，具有特定的约束条件。

(2) 项目是在一定的组织结构内部进行，利用有限的人力、物力以及财力等资源，在规定的时间内完成。

(3) 项目要满足一定的数量、质量、功能、性能和技术指标等多方面的要求。

项目是一种特殊的任务，它的执行过程与其他任务有着很大的区别。其主要特征包括

以下几个方面：

（1）项目的唯一性。每一个项目都有着自身独特之处，表现在目标、环境、条件、组织、过程等诸多方面，没有两个完全相同的项目。因此，尤其是在有风险存在的情况下，项目是不能够完全程序化的。

（2）项目的一次性。项目是一次性任务，一旦完成，就宣告这一项目结束，这是项目与其他重复性工作的最大区别。项目的一次性特性是针对项目整体而言，它是由为实现目标而开展的一系列活动的有机组合而形成的一个完整过程。

（3）项目目标的明确性。项目是一类特殊任务，它有着明确的目标。项目的目标包括成果性目标和约束性目标。成果性目标是由一系列技术指标如时间、费用、性能、功能等来定义的，约束性目标是项目实施过程中必须遵守的条件。项目目标是成果性目标和约束性目标的统一。

（4）项目结果的不确定性。在定义项目时，有时很难定义项目的目标、估算所需要的时间和经费；项目在进行过程中可能会有难以预见的技术、规模方面的问题；以及软件项目开发人员所存在的流动性，这些都会给项目的开发带来一定的风险，因此软件项目的运作存在着较大的不确定性和风险性。

（5）项目资源的消耗性。在整个项目的研发过程中，都会用到各种各样的资源。通常完成软件项目所需要的资源包括办公环境、人力资源、研发经费、硬件设备、网络环境、操作系统、开发工具、支撑软件等。这些资源有的是一次性消耗，也有的是可以重复使用。

项目通常由以下五个基本要素构成。

- 项目的范围：包括项目的内容、目标以及要求。
- 项目的组织：包括项目的团队组织以及管理模式。
- 项目的费用：包括项目的成本计划以及成本核算。
- 项目的质量：包括项目的质量标准和交付成果。
- 项目的进度：包括项目的进度计划和执行控制。

在项目目标的五个基本要素中，项目的范围与组织是最基本的，也是项目的核心问题；而项目的费用、质量、进度等是可以变动的，是依附于项目的范围和组织的。

3.1.2　软件项目的特点

软件项目是一种特殊的项目，通常是指采用某种计算机编程语言，为实现一个目标系统（即软件产品）而开展的活动，其目的是实现各类业务系统的信息化、业务管理的集成化与业务执行的连续化。

软件项目除了具有一般项目的基本特征外，还具有以下特征：

（1）软件项目是知识密集型项目。软件项目具有技术性强、多学科知识互相渗透的特点。软件项目的管理涉及多方面的知识，包含系统工程学、统计学、心理学、社会学以及法律等范畴的知识，这也正是软件项目区别于其他实体项目的关键点之一。

（2）软件项目采用以用户为中心的理念。用户的满意度是衡量现代软件产品质量的根本指标，也是软件项目运作的宗旨。在具体实施时，体现在软件的共性和软件的个性两个方面。共性化需求是指能够支持软件系统运行的各种功能和性能指标；个性化需求是指要适应目标用户的使用偏好，这是一个软件项目在与同类软件项目竞争时取胜的重要因素。

(3) 软件项目的风险较大。软件项目由于其技术的高度复杂性和需求的不稳定性等因素,造成软件项目的风险控制难度相对较大,成功率相对较低。但是一旦某个软件产品开发成功,将会带来高额的回报率。

(4) 软件项目的管理严格。软件项目需要对整个项目过程进行严格地、科学地管理,尤其是对大型、复杂的软件项目而言。质量产生于过程,没有严格的过程管理,开发人员的个人能力再强也是无济于事的。

(5) 软件产品需要多次完善。任何一个软件系统或者软件产品,都不可能是一次完成并且永久使用的。随着信息技术的发展,计算机软件和硬件的更新速度非常快,使用软件的人员水平也在不断提高,这些都对软件系统提出了更高的要求。因此,软件系统是一个需要不断完善、不断改进的过程性产品。

(6) 软件项目的文档编写量较大。在软件项目的整个开发过程中,所涉及的文档种类和数量比较多,而且需要经常进行修改,文档资料的编写工作量在整个项目过程中占据了很大的比重,它是项目管理中十分重要的组成部分。

软件项目管理是一个庞大的系统工程,它是为了使软件项目能够按照预定的成本、进度、质量顺利完成,而对成本、人员、进度、质量、风险等进行的分析和管理的活动。

软件项目管理的主要目的是为了让软件项目的整个生命周期,从分析、设计、编码、测试到维护的全过程,都能在管理者的控制下,以预定成本、按照规定的期限和质量完成软件产品并且交付用户使用。

软件项目管理的主要内容包括人员的组织与管理、软件项目的计划、软件风险管理、软件配置与质量管理、软件过程能力评估等。具体地说,就是按照需求界定目标,即根据用户的具体需求确定软件项目的范围与目标;按照目标制订计划,主要包括分解目标、制订阶段性计划、制订各个阶段的资金和资源的配置方案;按照计划组织资源,执行管理过程,其中包括人力资源、设备资源、资金等的组织及分配;按照目标落实和考核阶段性成果;最后按照程序进行评估、分析、总结、改进和完善。

可以说,需求是依据,计划是前提,资源是保障,组织是手段,管理是核心,落实执行是保障,评估分析是监控。

3.1.3 软件项目的立项

软件项目一般分为委托开发项目和自主开发项目两大类。委托开发项目是用户为实现某一特定目标而委托软件开发单位所完成的软件项目开发,它又分为公开招标项目和定向委托项目两种。委托开发项目一般以招标、投标的形式开始。自主开发项目是软件开发单位根据市场需求或科学研究需要而开发的具有自主知识产权的软件项目。

1. 软件项目的立项过程

软件项目立项的关键环节是可行性论证。对于任何一个新的软件项目的提出,首先要进行项目论证。项目论证是指对可能实施的项目在技术上的先进性、可行性,经济上的可承受性、合理性,实施上的可能性、风险性以及使用上的可操作性、功效性等进行全面的、科学的综合分析,为项目决策提供客观依据的一种活动。通过对可能实施的项目的可行性进行研究、分析,可完成项目的立项过程。

软件项目立项一般需要经过下面几个过程：

（1）在发起一个软件项目时，项目发起人为了寻求有关方面的支持，使其了解项目的必要性和可能性，需要将发起项目的理由以书面材料的形式递交给该项目的支持者和相关领导，这种书面材料被称为软件项目立项建议书。这一阶段就是软件项目的发起阶段。

（2）软件项目提出之后，要对项目进行可行性研究分析，包括对现有系统的分析、对新系统的描述、对可选择的系统方案、投资和效益分析以及社会因素的影响等方面。在项目可行性研究结果表明该项目是可行时，项目才可以开始。如果可行性分析做得不好，有可能使项目无法实现预期的效果。这一阶段就是项目的论证阶段。

（3）项目经过论证、最终确认可行之后，还需要上报给上级主管领导或者主管部门，以获得对项目的进一步核准，同时也要获得上级主管领导和主管部门的支持和帮助。这一阶段就是项目的审核阶段。

（4）在完成了项目的需求评估、可行性研究以及其他分析论证之后，经过上级主管部门的批准，就将项目列入到项目计划中。这一过程就叫做项目的立项。

项目立项完成后，就可以开始组建项目团队，进行项目的实施。

2. 软件项目可行性研究的任务

从上面的讨论中可以看出，软件项目立项的关键环节就是可行性论证工作。可行性研究的主要目的是回答问题“此项目是可以做的，还是不可以做的”。通常应从以下四个方面研究软件开发方案的可行性：

（1）技术可行性研究。技术可行性是根据用户提出的系统功能、性能以及实现系统的各项约束条件，来确定使用现有的技术是否能够实现这个系统。

（2）经济可行性分析。经济可行性是通过成本-效益分析，进行软件系统开发成本的估算，对软件系统成功后可能取得的效益进行估算，确定拟开发的软件项目是否值得投资开发。

（3）操作可行性分析。操作可行性是指新的软件系统在给定的工作环境中能否顺利运行，现有的管理制度、人员素质和操作方式等是否与新的软件系统兼容。

（4）法律可行性分析。法律可行性是分析在软件系统开发过程中可能出现的涉及法律的问题，是否会侵犯第三方的利益，是否会违反国家的法律。

3. 软件项目可行性研究的步骤

软件项目的可行性的研究阶段从系统目标与规模说明开始，直至提出新系统的可行推荐方案。一般包括以下几个步骤：

（1）明确新的软件系统的目标和规模。在这一阶段，系统分析人员仔细分析现有的资料，进一步了解项目的目标和规模，着重弄清楚用户需要解决的问题，然后清晰地描述系统开发的限制和约束。

（2）研究现有的软件系统的结构。实地考察现有的软件系统，收集、研究和分析现有系统的文档资料和使用手册，重点了解现有系统可以做些什么、不可以做什么、现有系统的成本代价以及现有系统的对外接口等问题，这些都是设计新的软件系统的重要支持。

（3）导出新的软件系统的逻辑模型。根据对现有系统的分析研究，明确新系统的功能、处理流程和所受的约束，采用数据流图和数据字典描述数据在新系统中的流动和处理过程，

建立新系统的逻辑模型。

(4) 总结评价各种解决方案。系统分析人员根据新的软件系统的逻辑模型,从技术角度出发,根据用户的要求和开发的技术实力,导出若干个不同的物理实现方案并进行比较和评估。

(5) 选择最优方案制定设计计划。根据上述分析结果,分析人员提出是否进行该软件项目开发的意见。若该软件项目值得开发,则应在几个备选方案中选择一个最优实现方案,并详细说明该方案可行的原因和理由,制定进度表和预算表。

(6) 编写可行性研究报告。将上述可行性研究过程的结果,编写成清晰的文档报告,提请用户和使用部门审查。最后将报告提交给决策者,以决定是否继续开发该软件项目。

4. 软件项目可行性研究报告

在进行了系统分析以及可行性研究之后,对软件项目的系统目标和范围有了一定的了解,并获得了新系统的几种可行的解决方案,在此基础上就可编写可行性研究报告了。一般可行性研究报告的内容如下所示。

可行性研究报告的模板

1. 引言
    1.1 问题的定义
    1.2 系统的实现环境
    1.3 系统的约束条件
2. 管理总结
    2.1 关键性问题
    2.2 相关的注释
    2.3 在管理方面的建议
    2.4 系统的影响
3. 方案选择
    3.1 选择系统的配置
    3.2 选择方案的标准
4. 推荐方案描述
    4.1 方案的简要说明
    4.2 各系统元素的可行性
5. 成本-效益分析
    5.1 成本效益表
    5.2 成本效益分析描述
6. 技术风险评价
    6.1 所需的技术列表
    6.2 技术的风险分析表
7. 方案进度计划
    7.1 方案的进度表
    7.2 进度协调方案

8. 有关法律问题
  8.1　所涉及的法律问题
  8.2　法律风险的分析
9. 其他问题
10. 结论性意见

3.1.4　软件立项文档

对于一般项目的建设，其筹建单位或者项目法人提出的建议文件，被称做项目立项建议书。项目立项建议书是根据国民经济的发展以及国内外市场等条件，对拟开发的项目提出的框架性的总体设想。同样，软件项目的开发也需要有项目立项建议书。项目立项建议书也被称为项目建议书。

项目建议书是产品构思、立项调查的最终结果，在撰写正式的项目立项建议书之前，开发人员首先要在宏观层面上搞清楚"开发什么"、"怎样开发"、"产生怎样的价值"等重大问题，这就是产品构思；而立项调查的目的是为产品构思和可行性分析提供充分的、有价值的信息。下面是项目立项建议书编制的主要内容。

项目立项建议书的模板

项目摘要（简述项目立项建议书的主要内容）
1. 项目立项的意义、必要性以及国内外发展现状及趋势
  1.1　项目提出的背景及依据
  1.2　同类技术国内外发展状况、开发水平、研究方向、技术需求、竞争情况调查、趋势预测分析，以及项目技术水平、所处地位的介绍
2. 项目的主要内容及总体目标、规格、具体考核指标
  2.1　项目的主要内容
  2.2　项目的总体目标
  2.3　主要技术指标、经济指标
3. 项目主要解决的技术关键点及创新点
  3.1　项目解决的技术要点
  3.2　项目技术的创新点
4. 项目的计划进度
5. 项目的前期工作情况
6. 项目所需的基本条件、现已具备的条件和缺乏的条件
  6.1　项目所需的基本条件
  6.2　项目已具备的条件
  6.3　项目缺乏的条件
7. 项目的风险分析
8. 项目的效益情况
9. 项目的成果应用
10. 经费预算及筹措方法
11. 结论

3.2 签订合同的方法与文档

在项目启动阶段，通常投资方（用户）与项目承接方（供应商）之间需要签订一个合同。合同是用户和供应商之间具有法律效力的协议，它明确规定了双方的权利与义务。在合同签订之前，通常有一个招标和投标的过程；合同签订之后，即进入项目的实施阶段。在项目实施过程中要检查合同的履行情况；在必要时要进行合同变更管理。在项目结束时，要对照合同对项目进行验收。

3.2.1 合同的基本概念

合同是用户和供应商为达成一个项目的目标及其他规定内容、明确双方相互的权利和义务关系而达成的协议文件，它具有法律效力。通常，合同是一个项目存在的标志。

软件项目合同属于技术合同。技术合同是在法人之间、法人和公民之间，以及公民之间以技术开发、技术转让、技术咨询和技术服务为内容的合同。

在软件项目的招标、投标结束之后，就进入到了合同管理阶段。合同管理是围绕着合同生命周期进行的，它分为五个阶段：合同准备、合同谈判、合同签署、合同履行、合同终止。其中，合同履行中的工作主要包括合同履行期间的跟踪管理和变更管理。

对合同履行情况的跟踪管理是指，在履行合同期间，对合同当事人应尽的职责情况进行检查，并及时处理合同履行过程中出现的问题，例如合同的违约、合同的争议等情况。当然，在履行合同期间，产品的质量是需要重点关注的内容。

合同管理过程如图 3-1 所示。

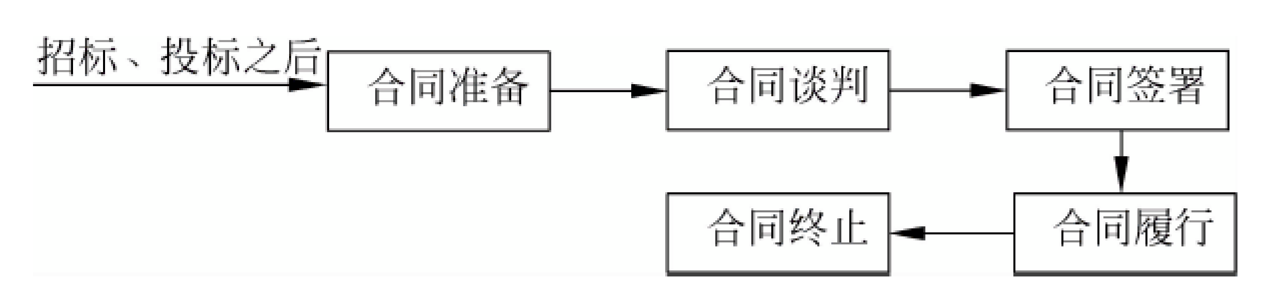


图 3-1 合同管理过程

签订合同之后，任何一方未经对方的同意，不得私自改动合同的内容，否则将被视为违约行为。如果在合同签订之后，双方当事人在履行合同的过程中，遇到了一些新的问题、新的情况，需要对项目的范围、双方的权利和义务以及一些其他方面的内容进行调整和重新规定时，就需要对合同进行一些修改或补充，这个过程被称为合同变更。

合同变更必须是经过双方当事人协商后，达成一致的意见，任何一方当事人未经对方同意而修改合同内容的行为都属于违约行为。

3.2.2 签订合同

我国《合同法》第三百二十三条规定，“订立技术合同，应当有利于科学技术的进步，加速科学技术成果的转化、应用和推广”。因此，订立和履行技术合同，作为法律行为的一种，应当符合一般法律行为的准则。

订立技术合同应当遵循以下基本原则：

(1) 遵守法律法规原则。

(2) 自愿公平诚实原则。

(3) 遵守社会公德原则。

(4) 促进科学技术成果转化推广原则。

合同有着各种形式。合同形式是指合同当事人设立、变更和终止民事权利和义务关系的方式。根据《合同法》第十条规定，“当事人订立合同，有书面形式、口头形式和其他形式”。

技术合同是较为复杂的交易活动，为了有利于合同的履行、监督、检查和管理，技术合同的订立、变更和解除一般采用书面形式。

签订技术合同的过程如下所示：

(1) 签订合同的双方当事人首先应当具备相应的资格，即应当具备相应的民事权利能力和民事行为能力。

(2) 双方当事人表达签订合同的意愿，以及包括合同成立所必须具备的条款，这就是要约。要约要表明一经承诺立即受到约束。

(3) 受要约人同意要约的意愿，这就是承诺。承诺必须由受要约人做出，承诺的内容必须与要约一致。

(4) 双方当事人在签字或者盖章时合同即成立。如果双方当事人没有同时在合同书上签字盖章，则以当事人中最后一方签字盖章的时间为合同成立时间。

3.2.3 合同的内容

合同内容是订立双方当事人意向的具体化表现，表现为合同的各项条款。根据我国《合同法》规定，在不违反法律规定的情况下，合同内容应由双方当事人共同约定。下面是一个软件项目的开发合同样本。

软件项目的开发合同样本

合同编号(0803)

甲方：	乙方：
地址：	地址：
邮编：	邮编：
电话：	电话：

签订地点：

公司网址：

上述甲、乙双方，经过友好协商，达成以下协议。双方声明，双方都已理解并认可本合同的所有内容，同意承担各自应承担的权利和义务，忠实履行本合同。

第一条 本合同中的技术开发项目的内容、工作进度与安排、数量、价款、交付和验收方式等由合同附件说明。

第二条 合同履行期限按照附件规定的工作进度决定，经双方协商一致，可以延长该期限(以下将其称为合同期限)。

第三条 甲方应向乙方提供必要的资料并负责与乙方联络、协调。

第四条 乙方承诺在履行合同期间，不进行有损甲方形象、声誉等的行为。

第五条 双方的基本权利和基本义务。

5.1 甲方的权利和义务

5.1.1 根据本合同项目的实际需要，以书面形式提供给乙方各项技术指标及功能

5.1.2 本合同标的使用应当符合国家法律规定和社会公共利益

5.1.3 对违反要求而进行的使用、操作所产生的影响、后果承担全部责任

5.1.4 按照本合同约定支付费用

5.2 乙方的权利和义务

5.2.1 严格按照甲方提出的各项技术指标、要求进行开发设计

5.2.2 根据甲方的要求举办培训和技术咨询

5.2.3 按照本合同约定收取费用

第六条 甲方同意按照双方约定的付款方式和时间，及时向乙方支付合同费用，以及提供其他必要的帮助。

第七条 甲方承诺，向乙方提供的内容、资料等不会侵犯任何第三方的权利；若发生侵犯第三方权利的情况，应由甲方承担全部责任。

第八条 乙方承诺，向甲方提供的软件系统必须是自行开发的，保证不是侵权软件；若发生侵犯第三方权利的情况，应由乙方承担全部责任。

第九条 乙方若不能按时提交软件系统，其责任由乙方承担。

第十条 本合同中的图文、程序、文件等版权属甲方所有。未经甲方许可，乙方不得公布、复制、传播、出售或者许可他人使用本合同中的图文、程序和文件等。

第十一条 甲方不能按时支付合同费用而导致的工期延误，其责任由甲方承担。

第十二条 双方当事人应当保守在履行本合同过程中获知的对方的商业秘密。

第十三条 双方应本着诚实信用的原则履行本合同。任何一方在履行中采用欺诈、胁迫或者暴力的手段，另一方可以解除本合同并有权要求对方赔偿损失。

第十四条 本合同签订后，经双方当事人协商一致，可以对本合同中有关条款进行变更或者补充，但应当以书面形式确认。上述文件一经签署，即具有法律效力并将成为本合同的有效组成部分。

第十五条 本合同附件为本合同不可分割的一部分，与合同正文具有同等法律效力。

第十六条 如果任意一方需要提前解除本合同，应提前通知对方。甲方提前解除合同的，无权要求乙方返还第六条的费用并应对乙方遭受的损失承担赔偿责任；乙方无故解除合同的，应双倍返还上述费用并应对甲方遭受的损失承担赔偿责任。

第十七条 任何一方没有行使其权利或没有就对方的违约行为采取任何行动，不应被视为是对权利的放弃或对追究违约责任或义务的放弃。任何一方放弃针对对方的任何权利，或放弃追究对方的任何过失，不应视为对任何其他权利或追究任何其他过失的放弃。

第十八条 任何一方违反本合同，给对方造成损失的，还应赔偿损失。在本合同与其他条款对违约有具体约定时，从其约定。

第十九条 因不可抗拒或者其他意外事件，或者使得本合同的履行不可能、不必要或者无意义的，任意一方均可以解除本合同。遭受不可抗拒、意外事件的一方全部或部分不能履行本合同、解除或迟延履行本合同的，应将事件情况以书面形式通知另一方并向另一方提交

相应的证明。

第二十条　订立本合同所依据的客观情况发生重大变化，致使本合同无法履行的，经双方协商同意，可以变更本合同相关内容或者终止合同的履行。

第二十一条　一方变更通信地址或者联系方式，应及时将变更后的地址、联系方式通知另一方，否则变更方应对此造成的一切后果承担责任。

第二十二条　双方当事人对本合同的订立、解释、履行、效力等发生争议的，应友好协商解决；协商不成的，双方同意向合同签订地的仲裁委员会提交仲裁并接受其仲裁规则。

第二十三条　本合同的订立、解释、履行、效力和争议的解决等均适用中华人民共和国法律。对本合同的理解与解释应根据原意并结合本合同目的进行。

第二十四条　如果本合同任何条款根据现行法律被确定为无效或无法实施，本合同的其他所有条款将继续有效。此种情况下，双方将以有效地约定替换该约定，且该有效约定应尽可能接近原约定和本合同相应的精神和宗旨。

第二十五条　本合同经双方授权代表签字并盖章，自签订日起生效。

第二十六条　本合同一式两份，双方当事人各执一份，具有同等法律效力。

甲方（盖章）	乙方（盖章）
授权代表签字	授权代表签字
年　月　日	年　月　日

附件：开发建设项目工作进度与安排、数量、价款、交付和验收方式。

1. 开发项目内容

（1）负责所需软件系统的开发、安装实施。

（2）对操作人员的培训。

（3）一年的维护服务。

2. 合同金额及付款方式

（1）本合同金额总计。

（2）付款方式。

合同签订两日内甲方向乙方支付合同总额的40%，即________；工程验收之后七日内甲方向乙方支付合同金额50%，即________；工程验收之后六个月内甲方向乙方一次支付合同金额10%，即________。

3. 完成时间及验收时间

（1）完成时间：本合同签订后________个月内制作完成。

（2）验收期限：甲方在接到乙方允许验收通知后________日内完成项目的验收，并确认。

4. 验收标准和验收后修改

（1）乙方所开发的软件符合甲方呈给乙方的书面要求及各项技术指标。

（2）乙方所开发的软件包含双方所确认的功能。

（3）验收期限为________天。

（4）验收合格，甲方应以书面方式签收，但甲方在乙方交付工作成果后一周内未书面签

收也未提出异议的，视为甲方验收合格。

(5) 验收合格后，如果甲方在使用过程中需要对工作成果进行修改，乙方可根据具体情况酌情优惠收取制作费。

(6) 软件的培训费用已包括在本软件开发合同书的合同金额内。

3.3 软件招标与投标

开发项目时的招标投标制度已经成为国际惯例，成为各国政府和企业所共同遵守的国际原则。随着软件行业的发展，招标、投标及评标制度在软件项目的开展过程中逐渐普及。从 2000 年 1 月 1 日起开始实施的《中华人民共和国招标投标法》使我国的招标投标过程变得有法可依。

3.3.1 项目招标与投标的基本概念

项目招标与投标过程的主要参与人有招标人和投标人。招标人是依据招标投标法的规定提出招标项目、进行招标的法人或者其他组织；投标人是指响应招标、参加投标竞争的法人或者其他组织。

所谓招标是指招标人直接向若干具有资质的法人或其他组织发放招标通知，或者采用招标公告方式，向不特定的人告知招标情况，以吸引投标人投标的行为。

所谓投标，是指投标人按照招标人的要求，在规定期限、规定的地点向招标人递送投标文档的行为。

项目的招标与投标具有以下特征。

(1) 公平性：招标与投标是独立法人之间的经济活动，必须本着平等、自愿、互利的原则和规范的程序进行，双方的权利和义务都受到法律的监督和保护。招标机构不得将各投标人区别对待，对各项投标的评审必须公平公正。

(2) 开放性：在公开招标中，招标机构要通过各种途径广泛告知有兴趣、有能力的投标人前来投标，进行自由竞争。此外，还要求招标机构对投标人说明交易规则、招标条件和最后结果，使之成为一种真正的开放性采购。

(3) 竞争性：招标与投标的核心是竞争。按照招标投标法的规定，每次招标必须有三家以上投标人参加投标以形成投标者的竞争。各投标人的目标是利用本身具有的优势以及竞争对方的弱点，通过各方面的努力，战胜其他投标者。

按照项目招标过程的特点，可以将招标分为公开招标和邀请招标。

1) 公开招标

公开招标是指项目招标人以招标公告的方式邀请不特定的法人或其他组织进行投标。公开招标的基本要求如下：

(1) 公开招标必须公开发布招标公告，通过国家指定的报刊、信息网络或者其他媒介如电视台等进行发布。

(2) 公开招标不得限制投标人的数量，任何对招标有意的、有能力的组织和个人都可以参加。

（3）公开招标必须以公开的形式进行开标，使投标人了解其他投标者的报价情况以及其他情况。

（4）公开招标在选择合适的中标人之后，要以一定的方式向中标人和其他投标人宣布投标结果。

2）邀请招标

邀请招标是指招标人以投标邀请书的方式在有限的范围内邀请一定数量的、特定的法人或者其他组织投标的行为。邀请招标的基本要求如下：

（1）在邀请招标中，招标通知不使用公开的广告形式，这主要是由于招标项目的特殊性。国家重点项目和省、直辖市人民政府确定的重点项目不适宜公开招标。

（2）在邀请招标中，只有收到邀请并接受邀请的人是合法的投标人，未收到邀请的组织和个人无权参加投标。

（3）在邀请招标中，投标人的数量是有限的，通常是具有承担项目开发能力、资信良好的三个或五个项目开发组织。

项目一般是通过招标、投标的形式开始。作为需求方的用户，首先根据自己对新项目的设想提出基本需求并编写招标书，然后将招标书通过公开招标或邀请招标的方式分发给有意向开发该项目的单位，即竞标方。按规定，竞标方通常要有三家或以上，以形成竞争关系。各竞标方在收到招标书后，会准备一份解决用户问题的方案，即标书。为了使自己的方案具有竞争力，竞标方在准备方案时会考虑在满足用户需求的基础上尽量降低费用，并附加上一些资质证明和所参加过项目的介绍，用以向用户强调自己的资历和能力。在若干满足用户需求的投标书中，通常用户方会根据标书报价选择一个最优的方案，提出该方案的投标人即中标方。然后中标方和用户通过进一步的讨论和研究、切磋，以最后确定项目开发合同的归属。竞标方获得中标以后，招标、投标工作就结束了。

3.3.2　软件招标与投标的过程

作为投资方（用户），软件项目招标的一般过程如下：

（1）首先撰写招标申请。

（2）进行招标资格的认定及备案。具有编制招标文件和组织评标能力的软件项目招标人可以自行办理招标，并按规定向行政主管部门备案；如果用户无法自己组织整个招标活动，也可以委托招标代理机构办理招标活动。这时委托方（即软件用户）应和招标代理机构签订委托代理合同。

（3）确定招标方式。按照相关法律法规和规章制度确定招标方式是公开招标还是邀请招标。

（4）拟定招标文件。招标文件主要包括项目招标公告、对招标单位的要求、投标人须知、招标章程、各种附件以及技术的相关资料。

（5）确定标底。标底是招标单位确定的价格底数，它是决定项目合同价格的主要因素，招标单位必须经过认真地测算确定标底。招标单位对标底必须绝对保密。

（6）发布招标公告。如果是邀请招标，则要拟定受邀单位（投标人）名单并送达招标邀请书。招标公告或招标邀请书是投标人进行投标的依据，其内容主要包括招标人的姓名和地址、招标项目的性质、招标项目实施的地点和时间、招标单位必备的条件、开标地点和时

间、招标文件的发售与价格等。

(7) 开标。软件项目招标机构应当按照招标书规定的开标时间和地点进行开标,招标机构对按时收到的标书进行开启并进行评标工作。

(8) 组建评标委员会。招标人依据法律法规和有关规章的规定,组建评标委会。通常评标委员会由招标机构的代表和有关技术、经济等方面的专家组成。

(9) 评标、定标。评标委员会必须本着公平、公正的态度对待所有投标人,对所有投标人的评价采用相同的程序和标准。评定内容主要从符合性鉴定、技术性鉴定、商务标评审、资格审查等几方面进行,然后推荐中标候选人或确定中标人。

(10) 招标结果公示。

(11) 发送中标通知书。招标人向中标人发出中标通知书,同时向未中标的投标人发出中标结果通知书。

(12) 签署合同及备案。为了管理和约束项目的委托和开发双方的权利和义务,以便更好地完成项目,招标人与中标人需要签署一个具有法律效益的合同,并向上级主管部门备案。

软件项目招标过程的示意图如图 3-2 所示。

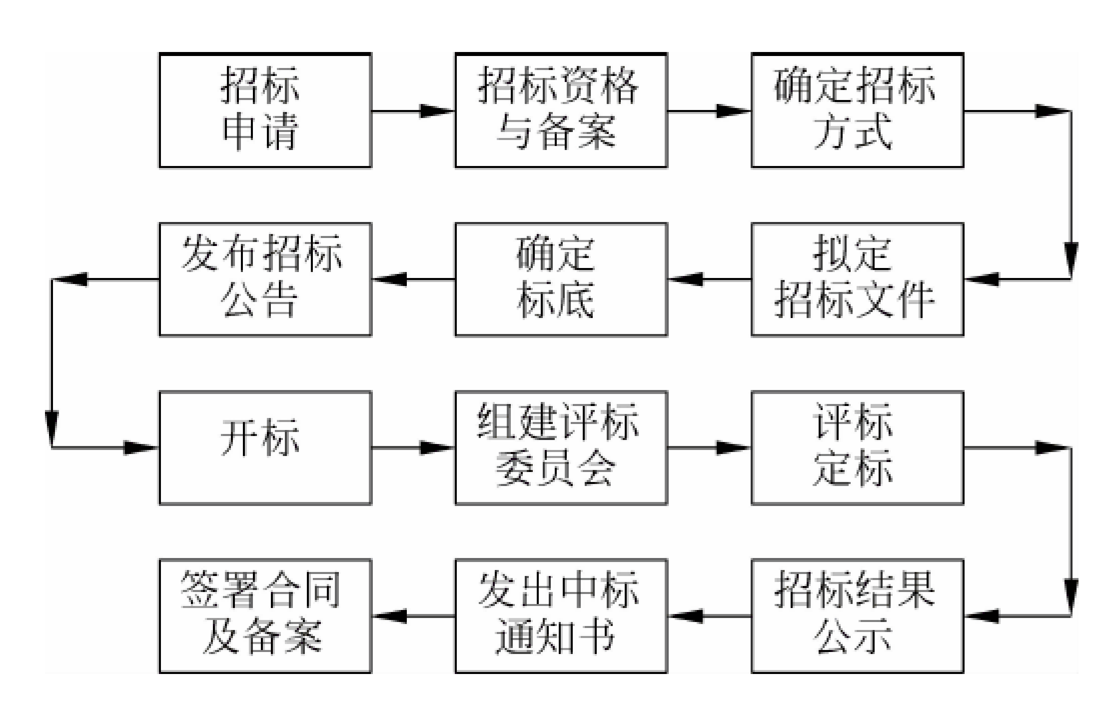


图 3-2 软件项目招标流程

实际上,在一般情况下,根据软件项目的实际情况不同,选择项目开发方的方式也有所不同,招标、投标是选择项目开发方最为普遍的一种方式,它是以一种严格的、公开的方式使软件项目的用户与开发方建立严密的、相互制约的合同关系,使得投资方(用户)能经济有效地组织资源,完成软件项目的实施。

3.3.3 软件招标书与投标书的编写

1. 软件项目招标书的具体内容

作为投资方(用户),在开始招标之前,首先要完成两个基础性工作:第一个是建立自己的招标机构或者选择代理招标机构;第二个是制定招标规则,编写项目招标书。

通常软件项目招标书的格式如下。

(1) 标题。招标书的标题由招标机构名称、招标项目名称和文书类别三部分构成。如《××大学教务处建立局域网招标书》。

(2) 正文。招标书的正文一般使用条文形式,也可以使用表格形式,主要包括三方面的

内容。

① 前言。简要说明本次招标的目的和依据、招标项目名称、招标范围等内容。

② 招标项目。这是招标书中的核心部分。这一部分具体写明本次招标的内容和要求、招标项目或产品名称以及规模、数量等，要让投标人全面了解招标单位所提供的各种有关信息。

③ 招标步骤。这一部分内容应该写明招标人的联系单位、地址、本次招标的起止日期、招标方式、开标时间及地点、投标截止日期等。

(3) 结尾。结尾部分详细写明招标单位的全称、地址、邮编、联系电话、电报挂号、电传号、传真号和联系人等。

招标书的详细程度和复杂程度随着招标项目和合同大小以及性质的不同而有所不同。但都必须有充分的资料，使投标人能够提交符合采购实体需求并使采购实体能以客观和公平方式进行比较的投标。

下面给出一个具体招标书的样书。

服装厂进销存软件开发招标书

××服装厂为了提高生产效率和管理人员业务素质，规范管理制度，计划于2012年9月1日之前实现全厂进、销、存业务的无纸化办公。为此向各软件开发单位公开招标。我们将本着公开竞争的原则，选择性能价格比最高的软件。现将我们对该系统功能的要求公布如下，欢迎各软件开发单位踊跃投标。

1. 物料管理

对我厂所有物料(包括各种主料、辅料、半成品、成品等)均采用等级式分类管理，并根据工厂实际规则实行统一编码管理，应保证物料操作中的通用性、准确性，以及易于使用的操作界面，方便操作员培训上岗。

2. 物料(成品)进出仓

可灵活方便地进行各种物料进出仓操作，提供入库收货单、退货单、领料单、退料单、调配单，及成品入仓单、成品出仓单等仓库物料进销存的各个操作模块。

所有的物料进出仓记录都应该与系统的订单、制作单及生产情况密切联系。管理人员应能随时查询订单物料的使用情况、损耗情况，并可结合员工工资等费用项目进行订单成本分析。

系统应能根据物料进出仓记录自动生成账簿式物料进出仓明细账，管理人员应可随时查看某一物料在指定时段内的进出仓日期、数量、金额及具体用在哪一张订单上。

3. 盘点/库存管理

可根据我厂实际情况分不同仓库录入所有物料(包括成品、半成品)盘点记录，并根据进出仓记录自动生成工厂最新库存数量，使管理人员随时都能知道厂内的物料库存情况，以便确定订单数量并及时进行生产安排。

应可随时对库存数据按各种所需条件进行过滤查询，或者分仓库、物料类别等方式进行报表打印，并可随时查看或打印物料的进销存情况，使用户随时掌握每一物料的使用情况。

4. 成本管理

根据订单物料采购、领用金额(系统根据物料用量及单价自动计算金额)及订单所用工资等费用自动算出生产成本，并对每张单的生产成本进行评估与分析，协助管理者对订单成本进行控制与调整。

5. 交货地点及时间

于 2012 年 9 月 1 日之前在××服装厂进行项目验收。

6. 售后服务

开发单位应根据双方签订的有关合同,提供技术培训以及三年以上软件更新售后服务。

7. 付款方式

付款方式按双方签订的有关合同执行。

2. 软件项目投标书的具体内容

软件开发单位如果有意参加一个招标软件项目的开发,就必须准备相应的投标书并参加投标。投标书是指投标人按照招标文件的条件和要求,在现场实地考察和调查的基础上所编制的文书,通常密封后邮寄或派专人送达招标机构,所以又称标函。

投标书是对招标文件提出的实质性要求和条件的响应和承诺。投标书的内容应当包括拟派出的项目负责人与主要技术人员的简历、业绩和用于完成招标项目的环境方案、技术开发方案等。软件项目的投标书应当将软件开发组织的历史业绩作为重要内容。

软件项目的投标书一般包括商务标部分和技术标部分。

商务标是明标,即应明确标示出投标人的身份,商务标的评标人根据给定的投标人信息对投标方的资质和实力进行评估。

技术标是暗标,即在标书的这一部分不能有任何能够确定投标人身份的信息,以便评标人能够仅对给出的技术方案本身进行评估,不应受到投标人身份的影响。

商务标部分主要是针对统一的商务报价而制定。要按照招标人提供的商务标格式编写。主要内容包括:

- 项目工作范围的说明;
- 项目工期、进度、质量、保修以及其他保证的说明;
- 软件项目各部分的报价和总报价;
- 相关的资质证明材料;
- 近几年来完成的软件项目一览表;
- 合理化建议。

技术标部分是对软件项目的技术处理方法和手段。技术标是暗标,投标人只能在标书封底规定之处填写招标机构名称、项目名称和投标人名称。在评标时,这些信息是加封的。技术标的具体内容应包括:

- 软件项目总体设计方案;
- 软件项目功能、性能和接口描述;
- 软件项目采用的技术环境;
- 投标人应当在招标文件要求的投标截止日期前,将投标文件送达投标地点。

3.4 下达任务的方法与文档

在软件项目开始实施阶段,由用户方或者项目双方共同制定项目开发任务书,作为整个开发工作的基础和依据。项目开发任务书中必须明确说明项目目标和项目范围两方面的

内容。

项目目标是实施项目所要达到的结果。在项目开发任务书中，项目目标的描述是一项非常重要的内容。项目目标的描述必须明确、具体、最好使用量化语言，以保证项目目标能够被正确理解。但是由于软件项目的特殊性，某些部分可能难以用非常清晰和具体的语言描述，但应尽可能地细化和达到可理解的程度。

项目范围是为实现项目目标所要完成的所有工作。正确地确定项目范围是保证项目成功的重要工作。如果项目范围界定不清楚，就有可能造成项目实施费用的增加、项目完成时间的拖延等情况，降低了劳动生产率，影响了项目按期按质地完成。

通常情况下，在与用户方签订了项目合同之后，就可以下达项目任务书了。项目任务书与项目合同同样重要，通常是项目的第二份管理文档。

下面是一份软件项目任务书的样本。

软件项目开发任务书样本

任务书名称：《××服装厂进销存业务管理系统》开发任务书。

下达日期：2012/03/01

发出部门：________________________

接受部门：________________________

1. 目标

开发目标定为行业通用进销存业务管理软件，而非仅针对××服装厂的具体业务要求。就是说，要有很高的产品化程度，以便于今后类似系统的开发和推广。因此，一切信息都要规范化、标准化、代码化。保证在产品实施时，其客户化工作只需要录入代码和修改代码，而不用对数据结构做出重大修改。

2. 功能模块划分及要求

进销存业务管理系统分为四个功能模块，要求每个功能模块具有高内聚、低耦合、信息隐蔽的性质，如表3-1所示。

表3-1　进销存业务管理系统功能

序　　号	模 块 名 称	功 能 要 求
1	物料管理	对物料进行等级式分类管理
2	进出仓管理	管理对物料和成品的进仓和出仓
3	盘点与库存管理	定期库存报表和随机查询
4	成本核算	根据物料采购、费用、工资等进行成本核算

3. 功能模块详述(略)

4. 功能模块任务分配

根据研发中心商业软件部目前的人员情况，本系统的项目经理由商业软件部副经理亲自担任，负责整个系统的规划、设计、协调与实施；商业软件部主任工程师担任产品经理，负责项目的整体需求、数据库设计与Alpha测试。整个项目分为四个任务组，各个任务组组长在项目实施阶段，承担小项目经理职责。四个任务组的人数及开发任务，如表3-2所示。

表 3-2　任务组的人数及开发任务

任　务　组	人　　数	具体开发任务
第 1 组	3	数据库结构设计与数据处理模块
第 2 组	6	用户界面设计及物料管理与进出仓管理模块
第 3 组	3	查询与报表模块
第 4 组	3	成本核算模块

5. 数据库与开发工具的选择

考虑到数据库的性能与价格比，数据库首选 MS SQL Server 2010。数据库设计工具采用 Power Designer，程序开发工具选择为.Net，文档制作工具为 MS Office 和 Power Designer。

6. 开发进度计划

研发中心软件部现有 15 人进入了本项目组。根据以往的实际工作经验，下面列出研发进度，如表 3-3 所示。

表 3-3　进度计划(2012/03/01～2012/09/01)

阶段名称	需求分析	概要设计	详细设计	编　码	测　试	培　训	验　收	备　注
第 1 周进度	需求培训							
第 2 周进度	需求获取							
第 3 周进度	需求获取							
第 4 周进度	需求确认							
第 5 周进度		概要设计						
第 6 周进度		概要设计						
第 7 周进度			详细设计					
第 8 周进度			详细设计					
第 9 周进度			详细设计					
第 10 周进度			详细设计					
第 11 周进度				编码				
第 12 周进度				编码				
第 13 周进度				编码				
第 14 周进度				编码				
第 15 周进度				编码				
第 16 周进度				编码				
第 17 周进度				编码				
第 18 周进度					Alpha			
第 19 周进度					Alpha			
第 20 周进度					Alpha			
第 21 周进度					Alpha			
第 22 周进度					Beta			
第 23 周进度						培训		
第 24 周进度						培训		
第 25 周进度							验收	
第 26 周进度								机动

7. 评审计划

各里程碑的评审计划，如表 3-4 所示。

表 3-4　里程碑评审计划

阶段名称	评审日期	评审地点	主持人	参加人	应 交 文 档
需求分析	2012/04/01	公司会议室	部门经理	项目组成员	用户需求报告/需求规格说明书
概要设计	2012/04/15	公司会议室	部门经理	项目组成员	概要设计说明书
详细设计	2012/05/15	公司会议室	项目经理	项目组成员	详细设计说明书
Alpha 测试	2012/07/10	公司会议室	项目经理	测试人员	Alpha 测试报告
Beta 测试	2012/08/01	客户单位	项目经理	客户代表	Beta 测试报告
验收	2012/08/22	客户单位	部门经理	验收组	验收报告

3.5　本章小结

在软件项目的开始阶段，必须要做好项目的立项工作，这是项目后续工作的基础。本章主要介绍了软件项目的立项、合同的签订以及招标与投标的有关内容与方法，重点介绍了相关文档的编写方法。

项目是指在一定的时间内，为满足某一特定目标所做的多项相关工作的总称。而软件项目又是一种特殊的项目，通常是指采用某种计算机编程语言，为实现一个目标系统而开展的活动，其目的是实现各类业务系统的信息化、业务流程管理的集成化与业务执行的连续化。

软件项目管理是一个庞大的系统工程，其主要目的是为了让软件项目的整个生命周期，从分析、设计、编码、测试到维护的全过程，都能在管理者的控制下，以预定成本、按照规定的期限和质量完成软件产品并且交付用户使用。

软件项目立项的关键环节是可行性论证，项目立项主要包括项目的发起阶段、项目的论证阶段、项目的审核阶段和项目的立项阶段。软件项目的开发需要有项目立项建议书。

合同是一个项目存在的标志。在项目启动阶段，通常投资方（用户）与项目承接方（供应商）之间需要签订一个合同，它明确规定了双方的权利与义务。合同内容是订立双方当事人意向的具体化表现，形式为合同的各项条款。合同管理分为五个阶段：合同准备、合同谈判、合同签署、合同履行、合同终止，其中在合同履行中主要包括合同履行期间的跟踪管理和变更管理。

项目一般是通过招标、投标的形式开始。招标投标制度已经成为国际惯例，成为各国政府和企业所共同遵守的国际原则。招标是指招标人直接向若干具有资质的法人或其他组织发放招标通知，或者采用招标公告方式，向不特定的人告知招标情况，以吸引投标人投标的行为。投标是指投标人按照招标人的要求，在规定期限、规定的地点向招标人递送投标文档的行为。

在与用户方签订了项目合同之后，项目双方共同制定项目开发任务书，作为整个开发工作的基础和依据。项目开发任务书与项目合同同样重要，必须明确说明项目目标和项目范围两方面的内容。

习题 3

1. 什么是项目？项目有哪些特点？
2. 软件项目立项的一般过程包括哪些？
3. 简述合同管理的过程。
4. 订立技术合同应当遵循哪些基本原则？
5. 如何确定软件项目的招标方式？
6. 简述软件项目的招标程序。
7. 软件项目招标与投标的主要特征有哪些？
8. 下达任务的时间和方法是什么？
9. 如何理解软件项目和产品的“功能、性能、接口”三项指标？
10. 请在教师的指导下，选择一个软件项目，编写一份《立项建议书》。

第4章 软件需求分析

随着现代社会的工业化、信息化，以及计算机应用技术的迅速发展，各种各样的软件系统随之被大量开发出来。软件开发人员面临着应用系统的复杂性越来越高，规模越来越大，而且支持系统开发的基础软件本身存在着复杂性、多样性、不间断性和自适应性等一系列问题。在这种状况下，软件需求分析的重要性和必然性就充分体现出来了。

软件需求分析是软件生命周期的一个重要阶段。只有通过软件需求分析，才能把软件功能和软件性能的总体概念描述为具体的软件需求规格说明，从而奠定软件开发的基础。软件需求分析质量的好坏直接影响着整个软件工程的进展和最终的结果。

4.1 需求分析的基本概念

软件工程所要解决的问题往往十分复杂，尤其是当建立一个全新的软件系统时，认识问题的本质是一个较为困难的过程。一般情况下，开发软件的技术人员精通计算机技术，但是并不熟悉用户的业务领域；而使用软件的用户虽然清楚自己的业务，但是并不掌握计算机技术。因此，通常对同一个问题，技术人员和用户之间可能存在着认识上的差异。面对这样的问题，在开始设计软件之前，就需要由既精通计算机技术、又熟悉用户应用领域的系统分析人员对软件方面的内容进行认真而细致地需求分析。

4.1.1 软件需求

IEEE(Institute of Electrical and Electronics Engineers，美国电气和电子工程师协会)软件工程标准词汇表(1997年)中将“需求”定义为：

- 用户为解决某一问题或者达到某个目标所需要的条件或能力。
- 系统或系统部件要满足合同、标准、规格说明以及其他正式规定的文档所需要的条件或者能力。
- 反映上面两方面的文档说明。

目前虽然对软件需求的定义有着不同的看法，但是通常认为软件需求是指软件系统必须满足的所有功能、性能和限制。软件需求分析是将用户对软件的一系列要求、想法转变为软件开发人员所需要的有关软件的技术说明。

在实际工作中，通常把软件需求细化为三个不同的层次，即业务需求、用户需求和功能需求。业务需求反映了组织机构或者用户对系统、产品的高层次目标要求，在项目视图与范

围文档中对它们进行了说明；用户需求描述了用户使用软件产品必须要完成的任务，这在使用实例文档或者方案脚本中进行了说明；功能需求定义了开发人员必须实现的软件功能，使得用户能够完成自己的工作，从而满足业务需求。图 4-1 描述了软件需求各组成部分之间的关系。

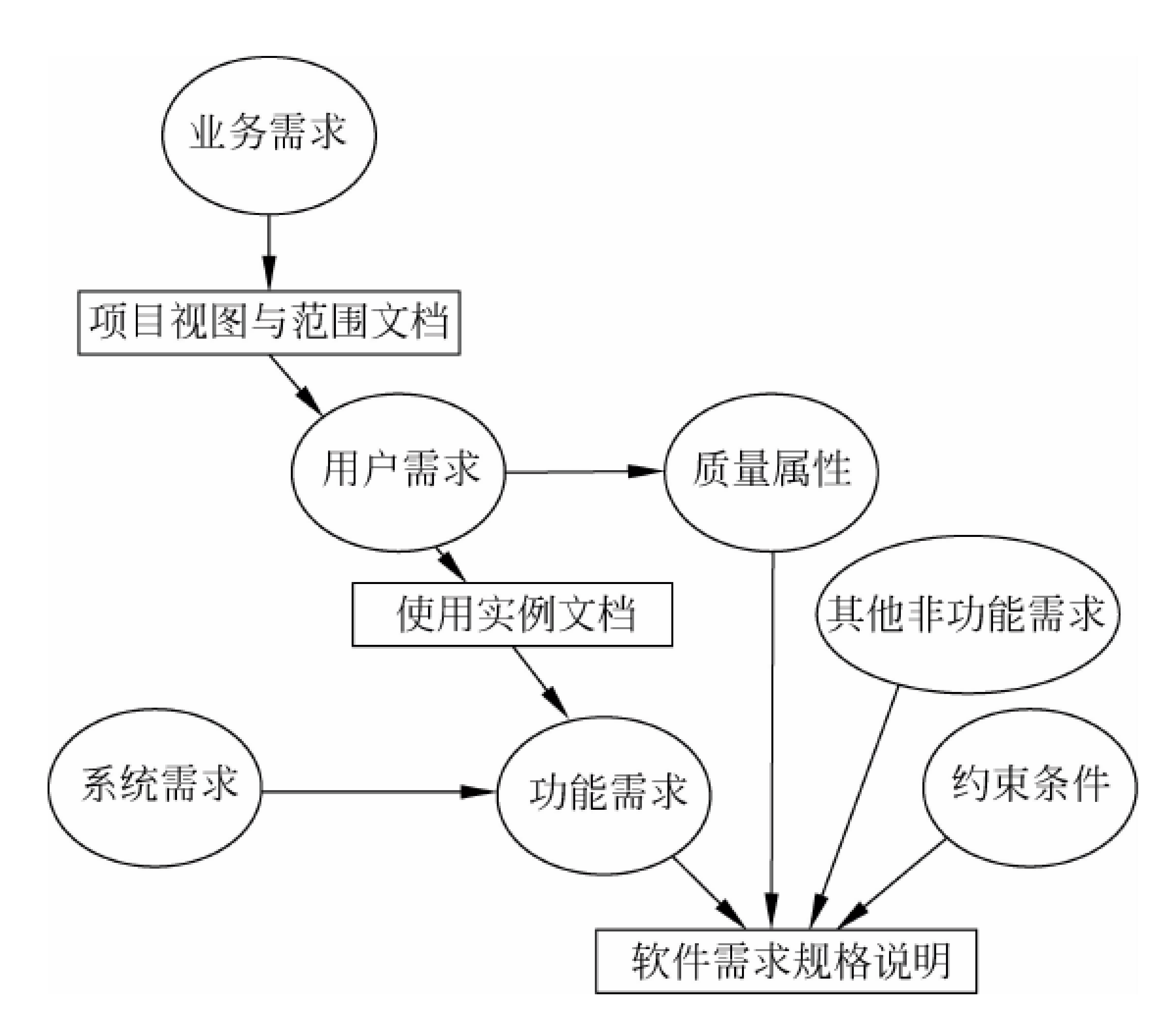


图 4-1 软件需求各组成部分之间的关系

1. 业务需求

业务需求是用来描述组织或用户的高层次目标，通常问题定义本身就是业务需求。业务需求必须具有业务导向性、可度量性、合理性以及可行性。这类需求通常来自于高层，如项目投资人、购买产品的客户、实际用户的管理者、市场营销部门或者产品策划部门。业务需求从总体上描述了为什么要开发这个系统，希望达到什么样的目标等一类问题，一般使用范围文档来记录业务需求，有时也被称为项目轮廓图或市场需求文档。

2. 用户需求

用户需求是用来描述用户使用软件产品必须要完成什么任务，怎么样完成。通常是在问题定义的基础上进行用户访谈和调查，对用户使用的场景进行整理，从而建立从用户角度出发的需求。用户需求必须能够体现软件系统将给用户带来的业务价值，或者用户要求系统必须能够完成的任务，即用户需求描述了用户能使用系统来做些什么。这个层次的需求是非常重要的。用例、特性等都是表达用户需求的有效途径。

3. 功能需求

功能需求是用来描述开发人员在产品中实现的软件功能，用户利用这些功能来完成任务，满足业务需求。功能需求是需求的主体，它描述的是开发人员如何设计具体的解决方案来实现这些需求。这些需求被记录在软件需求规格说明(Software Requirements Specification，SRS)中。软件需求规格说明完整地描述了软件系统的预期特性，一般可以是文档，还可以是包含需求信息的数据库或电子表格，或者是存储在商业需求管理工具中的信

息。开发、测试、质量保证、项目管理和其他相关的项目功能都要用到软件需求规格说明。

4.1.2　软件需求分析

在软件工程中，软件需求分析是指在建立一个新的软件系统或者改变一个现存的软件系统时，描述新系统的目的、范围、定义和功能时所做的全部工作。

软件需求分析是一项软件工程活动，它使系统分析人员能够描绘出系统的功能和性能，指明软件和其他系统元素的接口，并且建立系统必须满足的约束。通过对问题及其环境的理解与分析，对涉及的信息、功能以及系统行为建立模型，将用户的需求精确化、完整化，最终形成软件需求规格说明。

实际上，软件需求分析是对系统的理解与表达的过程。理解是指开发人员充分理解用户需求，对问题及环境的理解、分析和综合，逐步建立目标系统的模型。表达是指经过调查分析后建立模型，并在此基础上把分析的结果用规格说明等有关文档完全地、精确地表达出来。因此这一系列的活动就构成软件开发生命周期的需求分析阶段。

在软件工程发展的历史中，人们在很长一段时间里一直都认为需求分析是整个软件工程中最简单的一个步骤。但是，近年来越来越多的人认识到，需求分析是整个软件设计过程中最为关键和重要的环节。在进行软件需求分析时，如果开发人员不能正确掌握用户的真正要求，那么最后开发出的软件产品是不可能满足用户需求的。

4.1.3　软件需求分析的基本要求

软件需求分析是使用户需求具体化，并最终使需求满足用户要求。通常软件需求分析的基本要求包括以下几个方面：

- 完整性。在需求分析中，没有遗漏用户的任何一个必要的要求。
- 一致性。在需求分析中，用户和开发人员对于需求的理解应当是一致的。
- 现实性。需求应当是以现有的开发技术作为基础来实现的。
- 有效性。需求必须是正确且有效的，保证可以解决用户真正存在的问题。
- 可验证性。对于已经定义的需求是可以准确验证的。
- 可跟踪性。对于已经定义的功能、性能是可以被追溯到用户最初的需求。

4.1.4　软件需求分析的重要性

软件需求分析在软件开发过程中具有举足轻重的地位，它是开发出正确的、高质量的软件系统的重要保证。因此，无论是在学习软件工程的过程中，还是在开发软件产品的实践中，都要对软件需求分析有足够的重视。

软件需求分析在整个软件开发过程中的重要性主要表现为以下几个方面。

1. 软件需求分析是获得用户需求的有效途径

开发软件产品是为用户服务的，要想开发出真正满足用户需要的软件系统，首先必须掌握用户的需求。对软件需求的深入理解是软件产品开发工作获得成功的前提条件，否则如果开发出的软件产品不能真正满足用户需要，软件开发人员即使把设计工作和编程工作做

得再好也无济于事。

2. 软件需求分析是项目取得成功的关键因素

软件需求分析是一个项目的开始,也是项目建设的基础。在很多失败的项目中,大部分原因是由于项目需求不明确所造成的。项目的整体风险表现在需求分析不明确、业务流程不合理等方面,造成用户不愿意使用所开发出的软件产品,或者很难使用所开发的软件产品,从而使得项目失败。

3. 软件需求分析是软件设计的坚实基础

软件需求分析过程实际上就是确定用户需求的过程。由于用户掌握自己的需求,但是却不懂得如何使用计算机技术来加以实现;而软件设计人员往往缺乏实际事物的运作过程和商业过程的技巧,这时可以通过系统分析人员缩短商业领域和计算机技术之间的距离。从掌握需求信息的用户那里获得可用数据,并且把它转化成可以使用的形式,从而形成下一阶段软件设计的依据。

4. 软件需求分析是软件质量保证的重要阶段

软件需求分析阶段是项目的开始阶段,同时也是软件质量控制的开始时期。在软件生命周期的每一个阶段,都要采用科学的管理方法和先进的技术手段,并且在每个阶段结束之前都要从技术和管理两个角度进行严格审查,待审查合格之后再开始下一阶段的工作。这就使得软件开发工程的整个过程以一种井然有序的方式进行,从而保证了软件的质量,提高了软件的可维护性。

4.2 软件需求分析的过程和任务

软件需求分析是软件生命周期中非常重要的环节。在完成可行性研究之后,如果软件系统的开发是可行的,就要在软件开发计划的基础上进行需求分析。实际上,软件需求分析是一个不断认识和逐步细化的过程。在这个过程中将软件开发计划中确定的范围逐步细化到可以详细定义的程度,然后分析和提出各种不同的问题,并且为这些问题找到有效的解决方法。

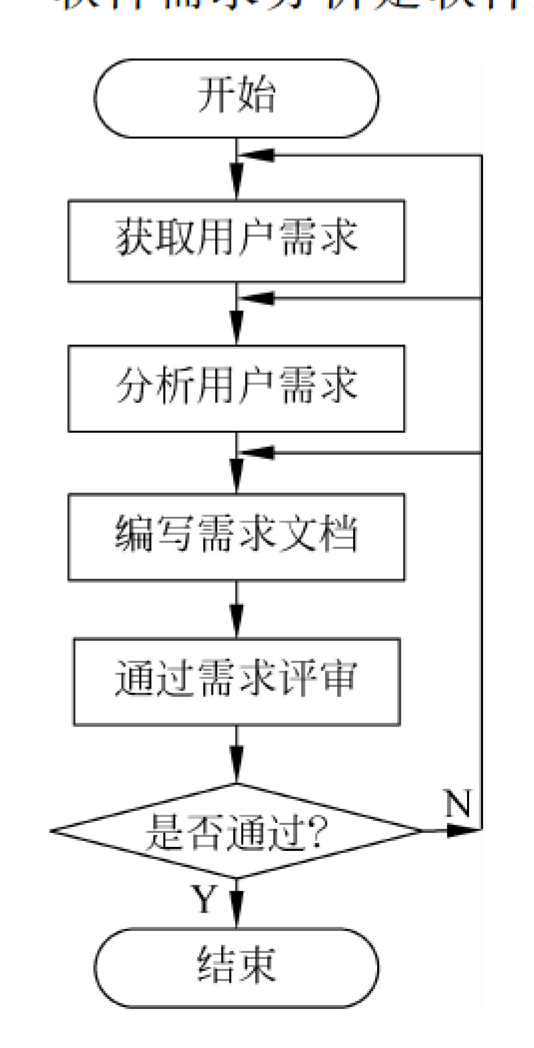


图 4-2 需求分析的过程

4.2.1 需求分析的过程

软件需求分析的过程主要包括获取用户需求、分析用户需求、编写需求文档以及通过需求评审等几个阶段,如图 4-2 所示。

1. 获取用户需求

在此阶段,必须充分地了解用户目标、业务内容、系统流

程，通过各种方式与用户进行广泛地交流，然后确定系统的整体目标和工作范围，弄清楚所有数据项的来源以及数据的流动情况。

2. 分析用户需求

在此阶段，分析人员从数据流和数据结构出发，根据功能需求、性能需求和环境需求，分析是否满足要求、是否合理，然后把其综合成系统的解决方案，给出目标系统的逻辑模型。分析和综合工作需要反复进行。

3. 编写需求文档

在此阶段，需要把已经确定的需求清晰、准确地描述出来，描述需求的文档被称为需求规格说明书。需求文档可以采用结构化语言编写文本型的文档，也可以建立图形化的模型，还可以使用数学上精确的形式化逻辑语言来定义需求。

4. 通过需求评审

需求分析直接关系到软件项目能否顺利，因此要求通过需求评审来控制需求分析的质量。需求评审可以通过内部评审、同行评审以及用户评审等方式进行。在需求分析评审中，用户的意见是第一位的。

4.2.2 获取用户需求的主要内容

软件需求分析的前提是准确、完整地获得用户需求。用户需求可以分为功能需求和性能需求两类。功能需求定义了系统应该做什么，系统要求输入哪些信息、输出哪些信息，以及如何将输入变换为输出；性能需求则定义了软件运行的状态特征，如系统运行效率、可靠性、安全性和可维护性等。

综合起来，应该获取的用户需求内容主要包括以下几个。

1. 物理环境

物理环境是指系统运行的设备地点以及位置有哪些，例如是集中式的还是分布式的；对环境的要求有哪些，例如对温度、湿度、电磁场干扰等的要求。

2. 软件系统界面

软件系统界面是指与其他系统进行数据交换时的内容与格式、用户对于界面的特定要求、用户在操作时的易接受性等。

3. 软件系统功能

软件系统功能是指系统主要能完成的任务，对于运行速度、响应时间或者数据吞吐量的要求，系统运行的权限规定，对可靠性的要求，对扩充性或者升级的要求等。

4. 数据要求

数据要求是指输入、输出数据的种类与格式，计算必须达到的精度，数据接收与发送的

频率,数据存储的容量和可靠性,数据或者文件访问的控制权限以及数据备份的要求等。

5. 文档规格

系统文档规格是指系统要求交付的各种文档,各类文档的编制规范,以及预期使用的对象等。

6. 维护要求

系统的维护要求是指当系统出错时,对错误修改的回归测试要求,系统运行的日志规格,以及可以允许的最大恢复时间的要求等。

4.2.3 需求分析的任务

软件需求分析要完成的任务就是要深入描述软件的功能和性能,确定软件设计的限制和软件与其他系统的接口细节,定义软件的其他有效性需求。即对目标系统实现的功能提出完整、准确、清晰、具体的要求。因此需求分析的任务就是借助于当前系统的逻辑模型导出目标系统的逻辑模型,重点解决目标系统"做什么"的问题。

软件需求分析的实现步骤如图 4-3 所示。

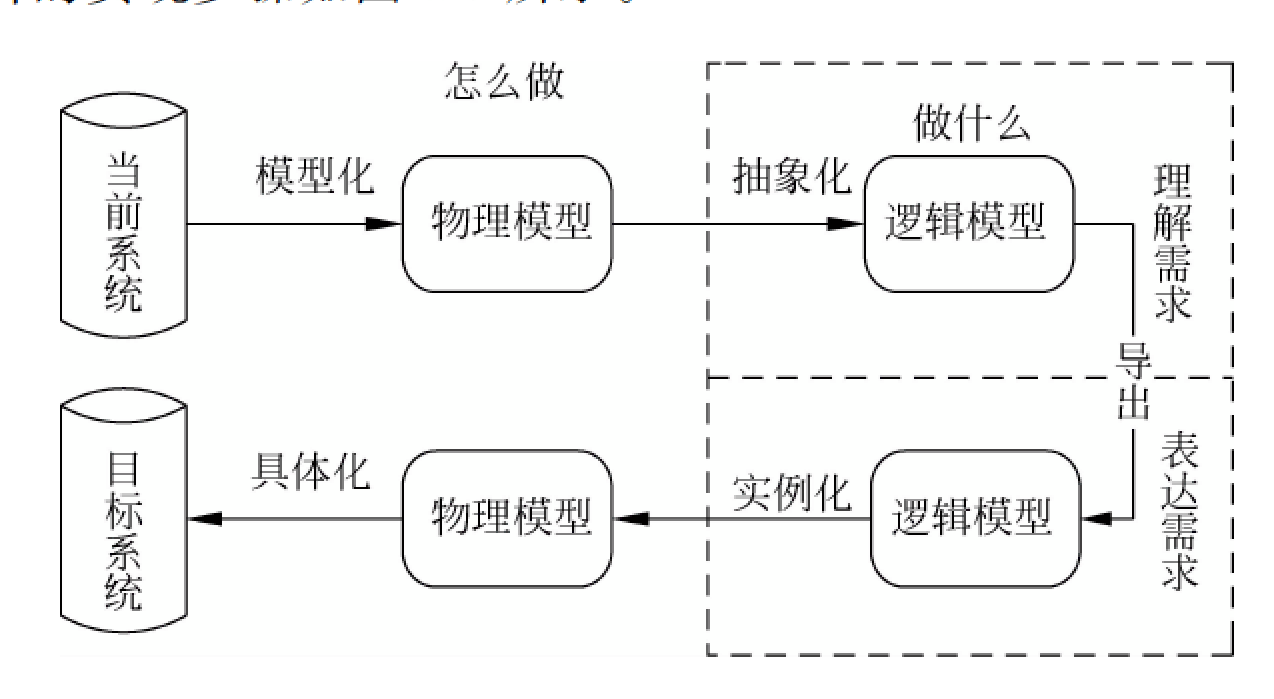


图 4-3 参考当前系统建立目标系统模型

由图 4-3 可知,软件需求分析的具体任务如下所示。

1. 确定对软件系统的要求

对软件系统的要求有以下四个方面:

(1) 系统的功能要求。功能要求就是划分出系统需要完成的所有功能。

(2) 系统的性能要求。性能要求包括联机系统的响应时间、系统需要的存储容量以及后援存储、重新启动、安全性等方面。

(3) 系统的运行要求。运行要求主要表现为系统运行时对环境的要求,如支持系统运行的系统软件是什么,采用的数据库管理系统是什么,数据通信接口是什么等。

(4) 将来可能提出的要求。应当明确列出那些现在不属于当前系统的开发范畴、但是根据分析将来可能会提出的要求,目的是在设计过程中对系统将来可能的扩充和修改做好准备工作。

2. 分析软件系统的数据要求

任何一个软件系统的本质都是信息处理系统，系统必须处理的信息和系统应该产生的信息在很大程度上决定了系统的构成。因此，必须分析系统的数据要求，这是软件需求分析的一个重要任务。通常分析系统的数据要求采用建立概念模型的方法。软件系统的数据要求就是归纳出目标系统的数据结构和数据之间的逻辑关系，描述系统所需要的输入和输出数据、数据库、数据类型以及数据的获取和处理方法等。

3. 导出软件系统的逻辑模型

在确定目标系统的要求和数据的基础上，通过一致性分析检查，逐步细化软件功能及各个子功能。同时对数据域进行分解直至分解到各个子功能上，以确定系统的构成，最后通过用数据流图、数据字典和主要的处理算法建立目标软件系统的逻辑模型。

4. 修正软件系统开发计划

根据在分析过程中获得的对软件系统的更深入、更具体的了解，可以准确地估计软件系统的成本和进度，从而修正以前制定的开发计划。

5. 开发原型系统

在计算机硬件和其他许多工程产品的设计过程中，经常使用原型系统。建造原型系统的主要目的是检验关键设计方案的正确性以及系统是否真正满足用户的需要。对于软件系统的开发，使用原型系统可以使用户通过实践获得对未来系统在运行时的更直接和更具体的认识，从而可以更准确地提出和确定用户的要求。

4.3 需求分析的方法

软件需求分析方法是由对软件问题的信息域和功能域的系统分析过程及其表示方法组成。信息域包括三种属性，信息流、信息内容和信息结构。需求分析方法有很多种，根据目标系统被分解的方式不同，基本上可以分为三种方法：20 世纪 70 年代开发出的“结构化分析方法”、20 世纪 90 年代初推出的“面向对象分析方法”和现今出现的“面向问题域分析方法”。

面向问题域分析（Problem Domain Oriented Analysis，PDOA）方法是一种比较新的技术，它更多地强调描述，较少地强调建模。其描述大致分为两部分，一部分注重问题域，另一部分注重解决系统的待求行为。在面向问题域的分析中，问题框架作为一个全新的模型被引用，该模型不仅有助于把需求从问题域的内在性质中划分出来，还有助于建立问题域的类型。现在人们对这种技术还没有全面地了解，并且缺乏文档资料。

4.3.1 结构化分析方法

结构化分析（Structured Analysis，SA）方法是 20 世纪 70 年代中期由 E. Yourdon 等人

提出的、适用于分析典型的数据处理系统的、以结构化方式进行系统定义的分析方法。这个方法通常与 L. Constantine 提出的结构化设计(Structured Design,SD)方法结合起来使用。该方法首先用结构化分析(SA)对软件系统进行需求分析,然后用结构化设计(SD)方法进行系统的总体设计,最后是进行系统的结构化编程(Structured Programming,SP)。

1. 结构化分析思想

结构化分析方法要求软件系统的开发工作按照规定的步骤,使用一定的图表工具,在结构化和模块化的基础上进行分析。结构化分析是把软件系统功能作为一个大模块,根据分析与设计的不同要求,进行模块分解或者组合。

在软件工程技术中,控制复杂性的两个基本手段就是“分解”和“抽象”。对于复杂问题,由于人们的理解力、记忆力有限,所以不可能触及到问题的所有方面以及全部细节。为了将复杂性降低到人们可以掌握的程度,可以把大问题分割成若干个小问题,然后分别解决,这就是“分解”。分解也可以分层进行,即首先考虑问题最本质的属性,暂时把细节忽略,以后再逐步添加细节,直至涉及最详细的内容,这就是“抽象”。

结构化分析方法的基本思路如图 4-4 所示。

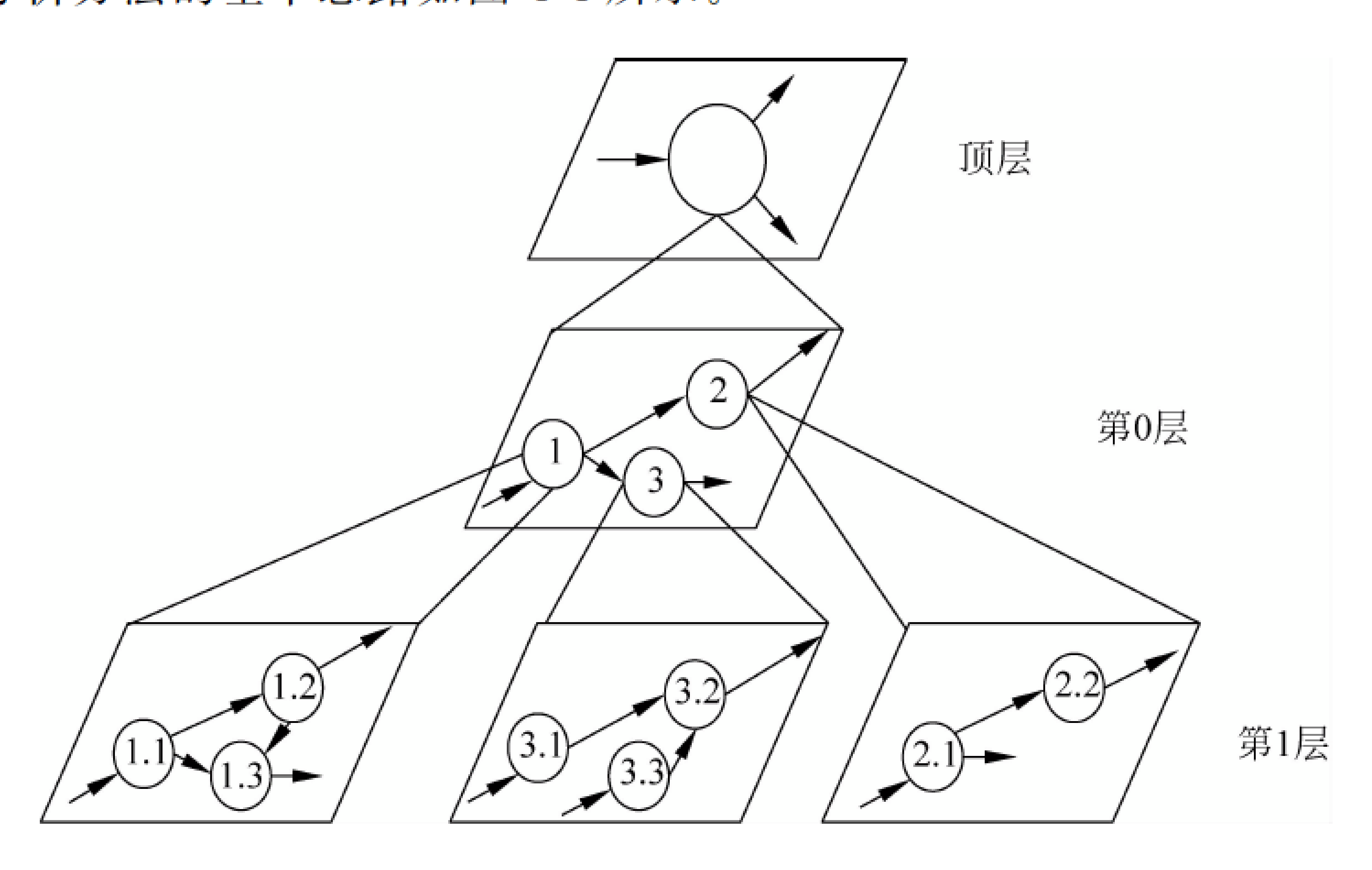


图 4-4　结构化分析方法的基本思路

这种方法使用了“自顶向下,逐步求精”的方式,使人们不至于一下陷入到细节部分,而是有控制地逐步了解更多的细节内容,这有助于理解问题。图 4-4 中的顶层抽象地描述了整个系统,第 1 层(底层)具体地表示了系统的每一个细节,第 0 层(中间层)则是从抽象到具体的逐步过渡过程。按照这种方法,无论问题多么复杂,分析工作都可以有计划、有步骤地进行。

2. 结构化分析步骤

采用结构化分析方法进行系统需求分析,一般包括以下步骤:

(1) 熟悉当前系统的工作流程,构造出当前系统的物理模型。当前系统是指目前正在运行的系统,也是需要改进的系统。通过对当前系统的详细调研,了解当前系统的工作过程,同时收集有关资料、数据、报表等,将所有收集到的信息用图形或文字描述出来,也就是

用物理模型来反映对当前系统的理解。

（2）分析当前系统的物理模型，抽象出当前系统的逻辑模型。当前系统的物理模型反映了系统“怎样做”的具体实现，要构造逻辑模型就要去掉物理模型中的非本质因素，保留本质因素。因为本质因素是系统固有的，是不随运行环境的变化而变化的。可构造当前系统的逻辑模型，以反映出当前系统“做什么”的功能。

（3）研究当前系统的逻辑模型，建立起目标系统的逻辑模型。目标系统是指拟开发的新系统。在当前系统逻辑模型的基础上，分析、比较目标系统与当前系统在逻辑上的差别，补充需要变化的部分，明确目标系统确实要“做什么”，这样就可以从当前系统的逻辑模型推导出目标系统的逻辑模型。

（4）进行目标系统的进一步补充和优化。为了对目标系统做完整地描述，还需要对逻辑模型做进一步补充和优化，其中包括要说明目标系统的人机界面，说明系统中的出错处理、启动与结束、输入输出和系统性能方面等需求的细节。对于系统特有的一些性能和限制，也需要用适当的形式做出书面记录。

分析阶段结束时，系统分析人员必须和用户再次认真审查系统文件，力争在系统开始设计之前，尽可能地发现其中还可能存在的错误，并且及时进行改正，直至用户确认这个模型确实表达了他们的需求之后，相关的系统文件才能作为用户和软件开发人员之间的“合同”而最终获得确定。

3. 结构化分析工具

结构化分析是一种建模活动，该方法使用简单易读的符号，根据系统内部数据的传递、变换关系，采用自顶向下、逐层分解的策略描述功能要求的软件模型。下面是使用结构化分析方法的一些指导性原则：

- 在开始建立分析模型之前，首先必须认真分析问题，切不可在问题没有被很好理解之前就产生一个带有错误问题的软件模型。
- 开发设计模型，使用户能够了解如何进行人机交互。
- 记录每一个需求的起源和原因，这样可以有效地保证需求的可追踪性。
- 使用多个需求分析视图，用来建立数据、功能和行为模型。
- 在需求分析中，应当尽量避免需求的含糊性和二义性。

结构化分析方法利用图形等半形式化的描述方法表达需求，形成需求规格说明书中的主要部分。常用的描述工具有以下几个。

- 数据流图：描述系统各部分组成以及各部分之间的联系。
- 数据字典：定义数据流图中每一个图形元素。
- 结构化语言、判断表、判断树：详细描述数据流图中不能被再分解的每一个加工。

由于分析中的主要依据是数据传递及数据变换所形成的数据流，所以结构化分析一般采用数据流图的分析方法，最终产生需求规格说明书。该文档包括一套数据流图、对数据流图中的成分进行定义的数据字典以及对加工逻辑的描述。

4. 结构化分析特点

结构化分析方法一般包括以下特点：

(1) 结构化分析方法是面向数据流的分析方法之一，它采用图形描述方法来建立分析模型，把软件系统描绘成一个可见模型，为系统的审查和评价提供了有力的条件，也为软件开发人员和用户提供了交换信息的方法，还为系统的设计阶段提供了坚实的依据。

(2) 结构化分析方法简单实用，特别适合于瀑布模型，易于开发者掌握，在成功率方面仅次于面向对象的方法。

(3) 结构化分析方法适合数据处理领域。为了使结构化分析方法适用于实时控制系统，还可以在数据流图中加入控制流，这是对结构化分析方法的一种扩充。

(4) 但是结构化分析方法不能提供对非功能需求的有效地理解和建模。

(5) 结构化分析方法通常会产生大量的文档，系统需求的要素会被隐藏在许多细节的描述中。

(6) 采用结构化分析方法建立的分析模型，只能是提供人们阅读的书面文档，而不能被机器阅读和运行。

(7) 结构化分析方法一般不容易被用户理解，因而很难验证模型的真实性。

4.3.2 面向对象分析方法

传统的结构化分析方法适合需求比较确定的应用领域，这已经成为软件工程界大多数学者的共识。实际上，系统的需求往往是变化的，而且用户对于系统需要也并不十分了解，但是这些问题在面向对象分析方法中已不再成为问题。面向对象方法是一种运用对象、类、继承、封装、聚合、消息传送、多态性等概念来构造系统的一种软件开发方法。

面向对象方法起源于面向对象程序设计语言，以后才逐渐形成了面向对象的分析方法和设计方法。

面向对象方法的某些概念可以追溯到 20 世纪 50 年代对人工智能的早期研究。但是人们一般把 20 世纪 60 年代由挪威计算中心开发的 Simula-67 语言看成是面向对象语言发展史上的第一个里程碑，它首次提出了对象的概念。Ada 语言是在 20 世纪 70 年代出现的又一种支持数据抽象的、基于对象概念的程序设计语言。

早期比较具有代表性和影响力的面向对象程序设计语言是由美国 Xerox(施乐)公司 Palo Alto 研究中心开发的 Smalltalk 语言。Smalltalk 全面实现了面向对象技术的机制，丰富了面向对象的概念，它的发布引起了人们对面向对象概念的广泛关注。

20 世纪 80 年代中期到 20 世纪 90 年代，是面向对象语言走向繁荣的阶段，产生了许多种面向对象的程序设计语言，如 C++ 和 Java 等。同时，面向对象的分析方法和设计方法也被广泛应用于软件的开发中。具有代表性的、基于面向对象思想的软件开发方法有 Grady Booch 提出的面向对象的分析与设计方法论、Jim Rumbaugh 提出的面向对象的建模技术和 Zvar Jacobson 提出的面向对象的软件工程方法学等。

面向对象的方法之所以如此流行，因为其本质是主张从客观世界固有的事物出发来构造系统，提倡人们用现实生活中的一般思维方式来认识、理解和描述客观事物，强调最终建立的系统能够映射问题域，即系统中的对象及对象之间的关系能够如实反映问题域中固有的事物及其关系。面向对象方法的主要特征如下：

(1) 认为客观世界是由各种对象组成的，复杂的对象是由简单的对象以某种方式组合而成。因此，面向对象的软件系统是由对象组成的，软件中的任何元素也都是对象，复杂的

软件对象是由简单的软件对象组合而成。

（2）把所有的对象都划分为各种对象类，每个对象类都定义了一组数据和一组方法，数据表示对象的静态属性，而方法则是对象所能执行的操作。因此在建立该对象类的一个新实例时，就可以按照类中对数据的定义为它生成一组专用数据，用以描述该对象独特的属性值。

（3）按照子类和父类的关系，把若干个对象类组成一个层次结构的系统（类等级）。在这种层次结构中，下层的派生类具有和上层的基类相同的特性，这就是继承。但是如果是在派生类中对一些特性做了修改，则在派生类中的这些特性以修改过的为准。

（4）对象与传统的数据有着本质的区别，对象之间只能够通过传递消息相互联系。也就是说，属于对象的各种信息（属性）和对象的行为（操作）都被封装在该对象类的定义中，在外面是看不见的，也不能直接使用，这就是封装性。

相对于传统的软件工程思想而言，面向对象的思想更符合人类的思维逻辑，它淡化了计算机的观点，以现实世界中的模型作为构造软件系统的依据。面向对象的基本概念包括对象、类、封装、继承和多态等。

4.3.3　统一建模语言

统一建模语言（UML）是一种标准的图形化建模语言，它主要用于软件的分析与设计。UML 的主要特点就是使开发人员用标准的、容易理解的方式建立起充分表达设计思想的系统模型和结构。UML 适合于各种软件开发方法、软件生命周期的各个阶段、各种应用领域以及各种开发工具。

1. UML 的发展

面向对象建模语言出现于 20 世纪 70 年代初期。随着面向对象的迅速发展，从 1989 年到 1994 年，面向对象建模语言的数量从不到十种猛增到了五十多种。在众多的建模语言中，语言的开发者努力推崇自己的产品，并在实践中不断完善自身。但是，由于面向对象方法的使用者并不了解各种建模语言的优缺点以及相互之间的差异，因而很难根据其应用特点选择合适的建模语言，于是爆发了一场“方法大战”。在这期间，最引人注目的是 Booch-93、OMT-2、OOSE 等方法。

Grady Booch 是面向对象方法最早的倡导者之一，他提出了面向对象软件工程的概念。1991 年，他把以前面向 Ada 的工作扩展到整个面向对象设计领域。Booch-93 比较适合于对系统软件的设计和构造。

Jim Rumbaugh 等人提出 OMT（Object Modeling Technique）方法，这种方法采用对象模型、动态模型、功能模型和用例模型完成对整个系统的建模，所定义的概念和符号可用于软件开发的分析、设计和实现的全过程。OMT-2 特别适合于分析和描述以数据为中心的信息系统。

Ivar Jacobson 于 1994 年提出了 OOSE（Object Oriented Software Engineering）方法，其最大特点是面向用例（Use Case），并且在用例的描述中引入了外部角色的概念。OOSE 方法比较适合支持商业工程和需求分析。

此外，还有 1991 年 Peter Coad 提出的 OOA（Object Oriented Analysis）/OOD（Object

Oriented Design)方法,它是最早的面向对象的分析和设计方法之一。该方法简单易学,适合于面向对象技术的初学者使用,但是由于这种方法在处理能力方面的局限性,目前已经很少被使用。

由于方法论的不同,对同一个问题的描述表示方法也有所不同,这样就在软件行业中的系统开发者之间产生了很大的混乱。

1994 年 10 月,Grady Booch 和 Jim Rumbaugh 开始致力于开发一种面向对象方法的标准。他们首先将 Booch-93 和 OMT-2 统一起来,并于 1995 年 10 月发布了第一个公开版本,称之为统一方法 UM(Unitied Method) 0.8。1995 年秋,OOSE 的创始人 Ivar Jacobson 加盟到这一工作中。经过 Grady Booch、Rumbaugh 和 Jacobson 三人的共同努力,于 1996 年 6 月和 10 月分别发布了两个新的版本,即 UML 0.9 和 UML 0.91,并将 UM 重新命名为 UML。

1996 年,一些机构将 UML 作为其商业策略,宣布支持并采用 UML。UML 的开发者们成立了 UML 成员协会,以完善、加强和促进 UML 的定义。这一机构对 UML 1.0(1997 年 1 月)及 UML 1.1(1997 年 11 月)的定义和发布起到了重要的促进作用。

1997 年 11 月 17 日,国际对象管理组织(Object Management Group,OMG)批准把 UML 1.1 作为基于面向对象技术的标准建模语言。2002 年 11 月推出了 UML 1.4 版,此后每隔几年就会进行版本更新,现在的版本是 UML 2.0 版。

UML 的发展历史如图 4-5 所示。

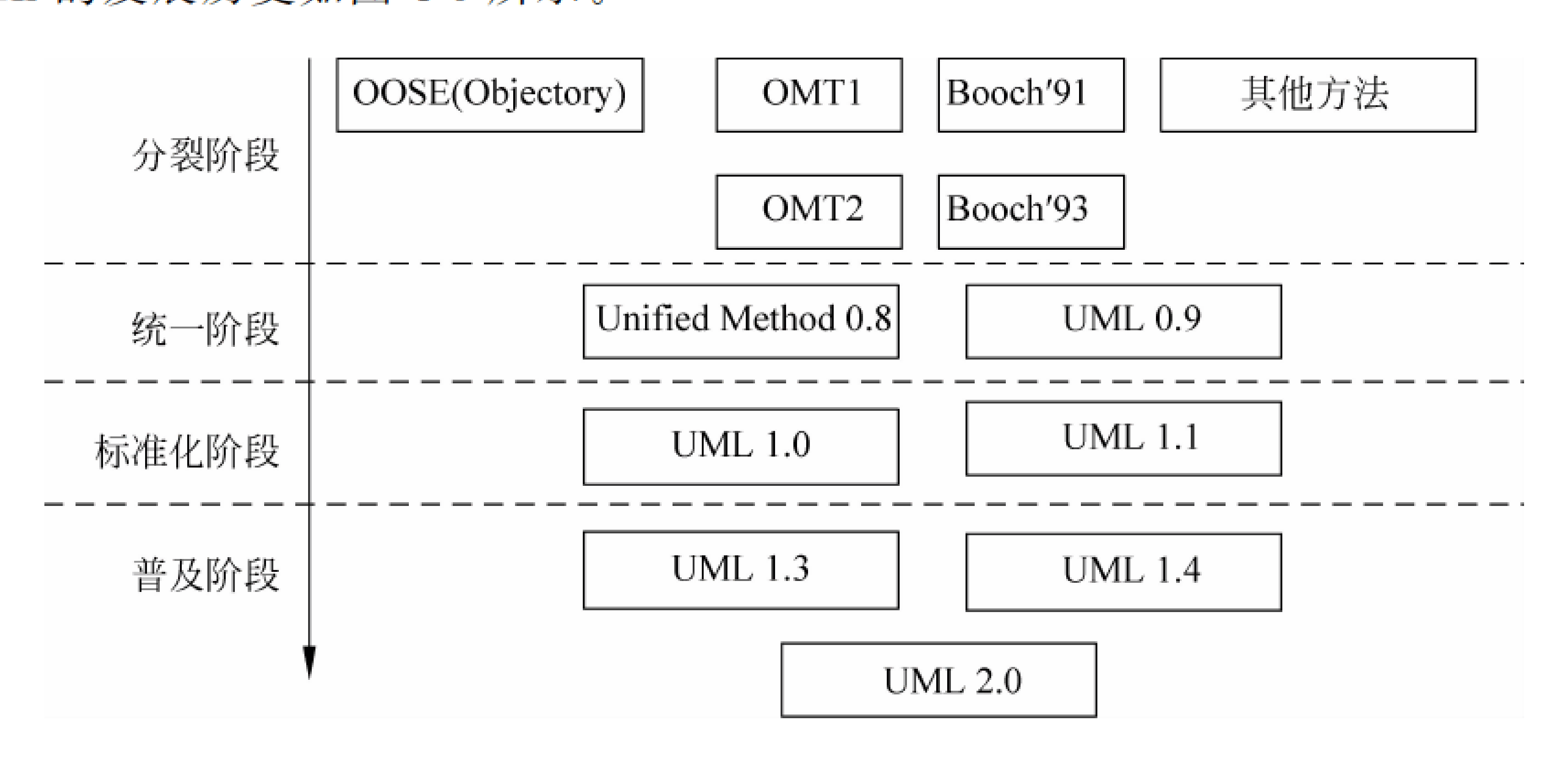


图 4-5 UML 的发展历史

2. UML 的基本概念

1) UML 的定义

Grady Booch 在他的经典论文 *The Unified Modeling Language User Guide* 中,对 UML 定义为:“UML 是对软件密集型系统中的产品进行可视化、详细化、构造化和文档化的语言。其中的产品是指在软件开发过程中的各个阶段所产生的各种各样的成果物,如模型、源代码、测试用例等。”

UML 代表了面向对象方法的软件开发技术的发展方向,在国际上受到越来越多的重视,它具有巨大的市场前景,也具有重大的经济价值和国防价值。

2）UML 的组成

作为一种建模语言，UML 的定义包括 UML 语义和 UML 表示法两个部分。

（1）UML 语义描述基于 UML 的精确元模型（Meta-Model）定义。元模型为 UML 的所有元素在语法和语义上提供了简单、通用的定义性说明，使开发者能在语义上取得一致，消除了因人而异的表达方法所造成的影响。此外，UML 还支持对元模型的扩展定义。

（2）UML 表示法用于定义 UML 符号的表示方法，为开发者或开发工具使用这些图形符号和文本语法进行系统建模提供了标准和规范。这些图形符号和文字所表达的是应用级的模型，在语义上它是 UML 元模型的实例。

3）UML 的特点

（1）UML 融合了 Booch、OMT 和 OOSE 等方法中的基本概念，而且这些基本概念与其他面向对象技术中的基本概念大多相同，因此 UML 必然成为许多方法使用者乐于采用的一种简单建模语言。

（2）UML 符号表示：综合了各种方法的图形表示，删除了容易引起混乱的和极少使用的符号，同时也添加了一些新的符号。因此，在 UML 中融汇了面向对象领域中的许多思想，这些思想是开发者们根据优秀的面向对象方法和丰富的计算机科学实践提炼而成的。

（3）UML 在演变过程中提出了一系列新的概念，在 UML 标准中加入了模板（Stereotypes）、职责（Responsibilities）、扩展机制（Extensibility Mechanisms）、线程（Threads）、过程（Processes）、分布式（Distribution）、并发（Concurrency）、模式（Patterns）、协作（Collaborations）、活动图（Activity Diagram）等新概念，并清晰地区分类型（Type）、类（Class）、实例（Instance）、细化（Refinement）、接口（Interfaces）、构件（Components）等概念。

4）UML 的应用

UML 是以面向对象图的方式来描述任何类型的系统，具有非常广泛的应用领域。其中最常用的是建立软件系统的模型，也可以用于描述其他领域的系统，如机械系统、企业机构或业务过程，以及处理复杂数据的信息系统、具有实时要求的工业系统或者工业过程等。

UML 适用于系统开发过程中从需求规格描述到系统完成后的测试的各个不同阶段。在需求分析阶段，可以使用用例来获得用户需求，通过用例建模，描述对系统相关的外部角色和对系统的功能要求。

UML 模型可以作为软件测试阶段的依据。系统通常需要经过单元测试、集成测试、系统测试和验收测试。不同的测试阶段使用不同的 UML 图作为测试依据，单元测试使用类图和类规格说明；集成测试使用部件图和合作图；系统测试使用用例图来验证系统的行为；验收测试由用户进行，以验证系统测试的结果是否满足在分析阶段确定的需求。

总之，UML 是一个通用的标准建模语言，可以对任何具有静态结构和动态行为的系统进行建模。

3. UML 的结构

UML 的结构包括以下几项：

（1）UML 的构造元素。UML 的构造元素包括三种类型，事物（Thing）、关系（Relationship）和图（Diagram）。

（2）UML 的事物。UML 的事物分为四种类型，结构事物、行为事物、组织事物和注释

事物。

- 结构事物(Structural Thing)。UML中的结构事物主要包括类(Class)、接口(Interface)、协作(Collaboration)、用例(Use Case)、活动类(Active Class)、构件(Component)和结点(Node)。
- 行为事物(Behavioral Thing)。UML中的行为事物包括交互(Interaction)和状态机(State Machine)。
- 组织事物(Grouping Thing)。UML中的组织事物只有一种,我们称之为包(Package)。包是一种有组织地将一系列元素分组的机制。
- 注释事物(An Notational Thing)。UML中的注释事物又称辅助事物,属于这一类的只有注释(Note)。注释是 UML 模型的解释部分。

(3) UML 的关系。UML 的关系可分为关联、依赖、泛化和实现,它们是 UML 中的基本关系构造模块。

- 关联:表示两个或多个对象之间的连接,是一种结构化关系。关联可能有方向,一般用一个实心三角形箭头来表示。
- 依赖:若一个对象 A 发生变化,可能会引起对另一个对象 B 的变化,则称 B 依赖于 A。在 UML 图中,依赖关系用一个带箭头的虚线表示。
- 泛化:表示对象的一般类和特殊类之间的关系。在面向对象中,这种关系被称为继承。泛化用带有空心箭头的直线表示。
- 实现:是将一种模型元素(如类)与另一种模型元素(如接口)连接起来,它表示不继承结构,只继承行为。实现用带有空心箭头的虚线表示。

(4) UML 的图。UML 使用模型来描述系统的结构及其行为。它从不同的视角为系统的架构建模,形成系统的不同视图(View),每一种视图都是由一个或多个图(Diagram)组成。UML 有四种类型的视图和九种图,如图 4-6 所示。

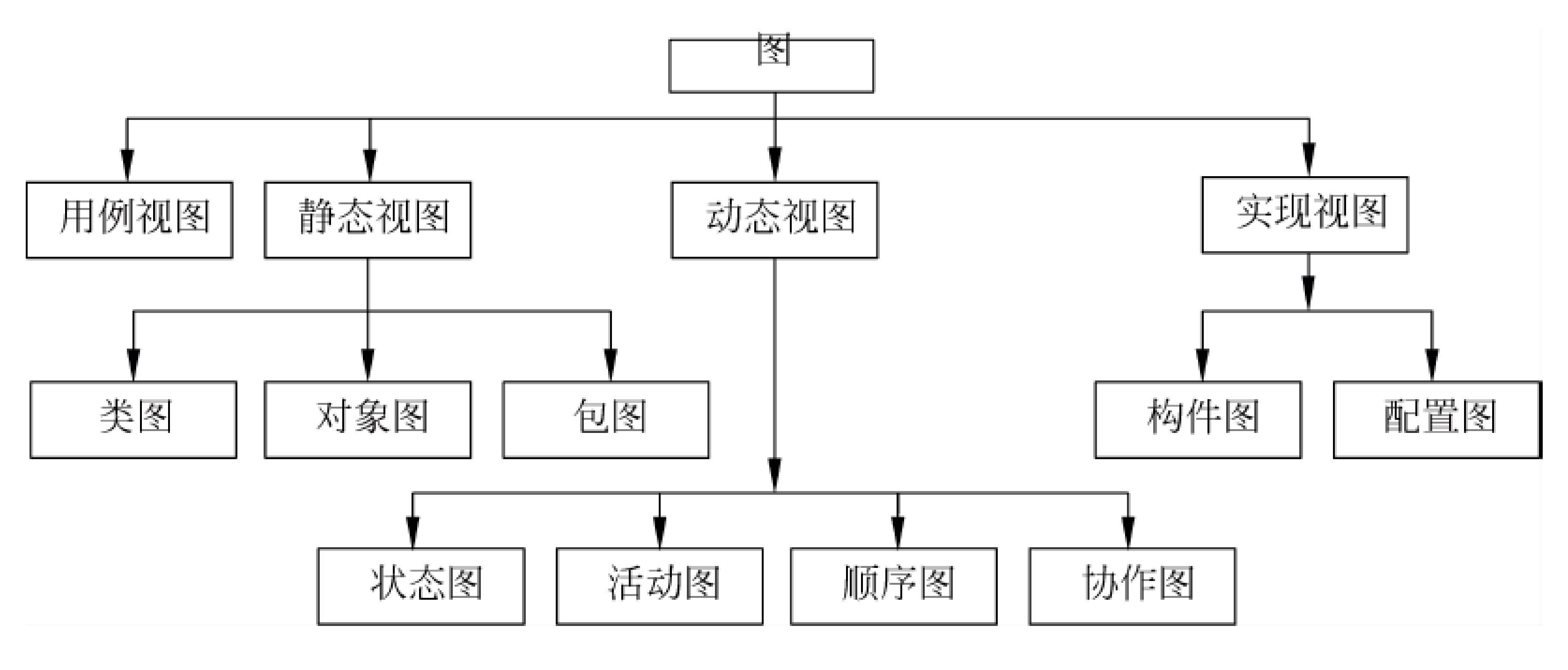


图 4-6 UML 的图

第一类是用例视图(Use Case Diagram)。用例视图表示角色和用例之间的关系,用于帮助用户和开发人员更好地理解所要开发系统的功能。在 UML 中,用例用一个椭圆表示,角色用人型符号表示。

第二类是静态视图 (Static Diagram)。静态视图包括类图、对象图和包图。

类图是描述类、接口、协作及其之间的关系,它描述的是一种静态关系,在系统的整个生

命周期都是有效的。类图不仅显示信息的结构，同时还描述了系统的行为。类在类图中用矩形框表示，其属性和操作分别列在分格中。

对象图是类图的一个实例，使用与类图基本上相同的标识。对象图显示某个时刻对象和对象之间的关系。由于对象存在生命周期，因此对象图只能在系统某一时间段存在。

包图是由包以及包与包之间的关系组成，用于描述系统的分层结构。包图的表示由两个长方形组成，其中小长方形位于大长方形左上角。如果不显示包的内容，则包的名字可以写在大长方形内，否则包的名字写在小长方形内。

第三类是动态视图(Dynamic Diagram)。动态视图描述系统的动态模型和组成对象间的交互关系，包括状态图、活动图、顺序图和协作图。

状态图是用于表示状态机(State Machine)的图，它由状态、转换、事件、活动等元素组成。状态图强调从一个状态到另一个状态的控制流。状态图由表示状态的结点和表示状态之间转换的带箭头的直线组成。

活动图是一种描述系统行为的图，它是强调计算过程中的顺序和并发步骤的状态机，表现系统内在对象间从一个活动到另一个活动的流程，类似于程序流程图。

顺序图描述对象之间的动态合作关系，它强调对象之间消息发送的时间顺序，同时显示对象之间的交互序列。顺序图有两维，垂直维代表时间，水平维代表对象。

协作图描述参加交互的各对象组织。协作图只对相互之间有交互作用的对象和这些对象之间的关系建模。如果强调时间和顺序，则使用顺序图；如果强调上下级关系，则选择协作图。

第四类是实现视图（Implementation Diagram）。实现视图包括构件图和配置图。

构件图描述了软件的各种构件及构件之间的依赖关系，通常包括构件、接口和关系。每一个构件图只描述系统实现中的一个方面，是系统代码的物理模块。

配置图描述系统中软件和硬件的物理结构。它可以显示结点以及结点之间的必要连接，可以显示这些连接的类型，还可以显示构件与构件之间的依赖关系。在 UML 中，一般用立方体表示结点，用实线表示关联关系。

当采用面向对象技术设计系统时，第一步首先要描述需求；第二步根据需求建立系统的静态模型，构造系统的结构；第三步描述系统的行为。其中，第一步与第二步中所建立的模型都属于静态模型，包括用例图、类图、对象图、包图、构件图、配置图等图形，是 UML 的静态建模机制。第三步中所建立的模型，或者可以执行，或者表示执行时的时序状态或交互关系，包括状态图、活动图、顺序图、协作图等图形，是 UML 的动态建模机制。因此，UML 的主要内容也可以归纳为静态建模机制和动态建模机制两大类。

4.4 需求描述工具

软件需求分析方法最初是作为人工使用而开发出来的，但是对于一些大型的软件项目，用人工方法进行分析比较困难。因此，针对这些方法，开发出了一些利用计算机的自动工具来帮助分析人员进行需求分析，这样在很大程度上改善了系统分析的质量、提高了系统分析的效率。

软件需求分析的自动工具按照其不同的表现形式分为两大类:

(1) 第一类主要是利用图形记号进行分析,产生一些图示,协助问题进行分解,自动生成和维护系统的规格说明。它的特点是将智能处理应用到问题的规格说明中。

(2) 第二类是一种特殊的、以自动方式处理的表示方法,用需求规格说明语言来描述需求,其特点是可以产生有关规格说明的一致性和组织方面的诊断报告。

软件需求分析的描述工具有数据流图、数据字典、结构化语言、判定表、判定树、层次方框图、Warnier 图、IPO 图、需求描述语言等。本节重点介绍前 5 种工具。

4.4.1 数据流图

数据流图(Data Flow Diagram,DFD)是一种从数据传递和加工的角度、以图形的方式描述数据流从输入到输出的移动变换过程。图 4-7 所示是一个简单的数据流图,它表示数据 X 从源 S 流出,经 P1 加工转换成 Y,接着经过 P2 加工转换为 Z,在加工过程中从 F 中读取数据。

在数据流图中,一般要用到四种基本符号,如图 4-8 所示。

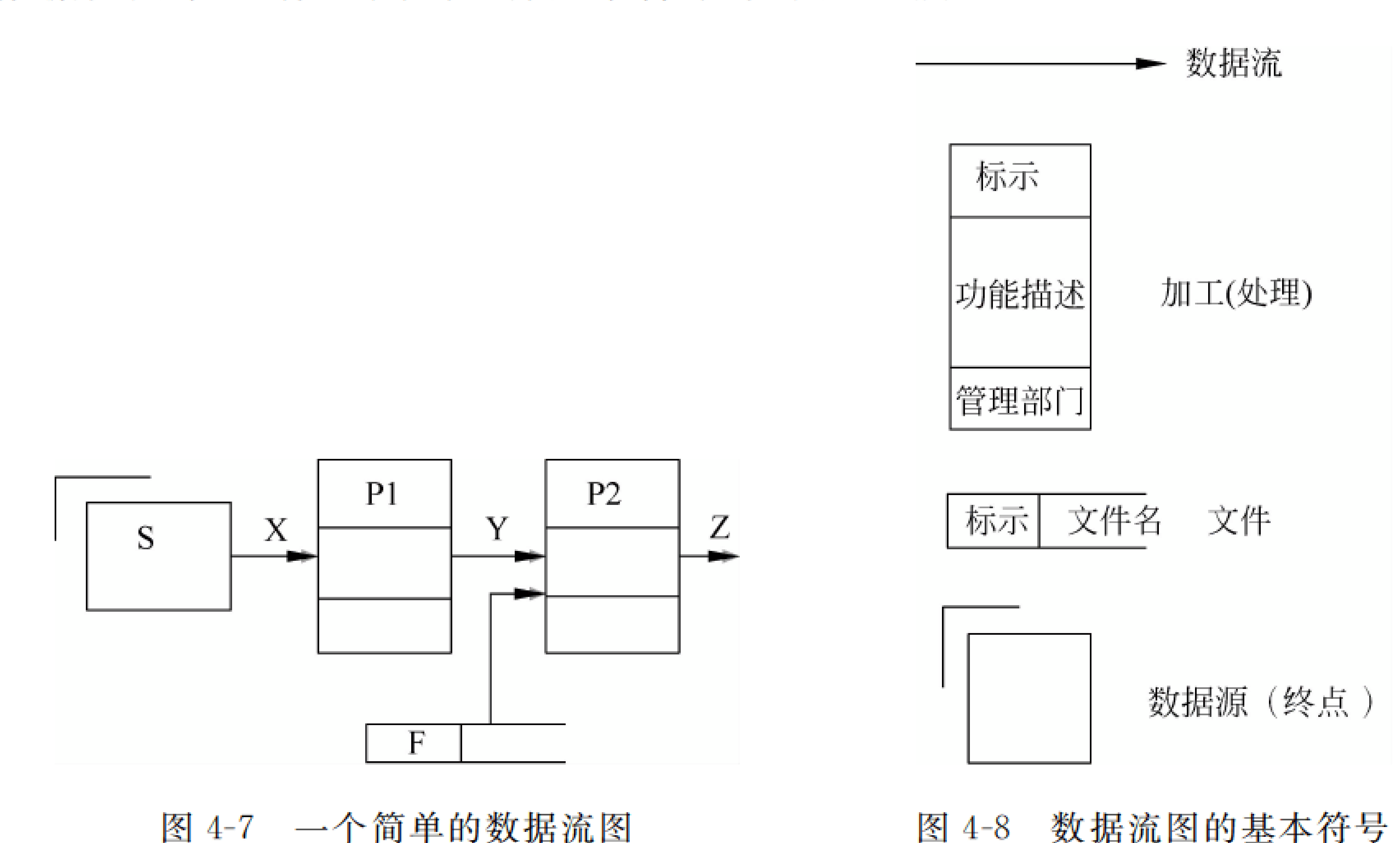


图 4-7 一个简单的数据流图　　图 4-8 数据流图的基本符号

1. 数据流

数据流是数据在系统中的传输路径,由一系列成分固定的数据项组成,其方向可以是从加工流向加工,从加工流向文件,或者是从文件流向加工。在数据流图中,数据流用带箭头的直线表示。

2. 加工(处理)

加工也被称为数据处理,用来表示对数据流进行某些加工或处理,是把输入数据转变成输出数据的一种变换,如对数据的算法分析和科学计算。每一个数据加工都应该有一个名字来概括其内容,用来表示其含义。

3. 文件

文件是存储数据的工具，表示数据的静态存储。数据可以存储在磁盘、磁带和其他存储介质中，必须对文件进行命名。文件名应与它的内容一致，写在开口长条内。在数据流图中，要注意指向数据文件的箭头方向。读数据的箭头是指向加工处理的，写数据的箭头是指向数据存储的，双向箭头表示既有读数据又有写数据。

4. 数据源（终点）

数据源或终点代表系统外部环境中的实体，可以是人、物或者其他软件系统。它们发出或接收系统的数据，使用起来并不严格，其作用是提供系统和外界环境之间关系的注释性说明，使得数据流图更加清晰。

在绘制数据流图的过程中，应当注意以下几点：

(1) 数据的处理可以是一个程序、一个模块，还可以是一个连贯的处理过程。

(2) 文件是指输入或者输出文件，它可以是文件、文件的一部分、数据库的元素或记录的一部分等。

(3) 数据流和文件是两种不同状态的数据。数据流是指流动状态的数据，而文件是指处于静止状态的数据。

(4) 当目标系统的规模比较大时，为了能清晰地描述和易于理解，通常采用逐层分解的方法，然后画出各分层的数据流图。在分解时，要注意考虑其自然性、均匀性、分解度等一些因素。

- 自然性是指概念上要合理、清晰。
- 均匀性是指把一个大问题尽量分解为规模均匀的几个部分。
- 分解度是指将问题分解的维度，一般应分解到基本加工为止。

(5) 对数据流图分层细化时，必须保持数据的连续性，即细化前后对应功能的输入数据和输出数据一定要相同。

4.4.2 数据字典

数据字典(Data Dictionary,DD)是软件需求分析阶段的另一个有力工具。数据流图描述了系统的分解过程，直观而且形象，但是没有对图中各个成分进行准确并且完整的定义。数据字典是为数据流图中的每一个数据流、文件、加工以及组成数据流或文件的数据项做出说明。

数据流图和数据字典一起构成了系统的逻辑模型。没有数据字典，数据流图就不严格；没有数据流图，数据字典也不起作用。在数据字典中，建立严格一致的定义有助于改进分析人员和用户之间的交流，避免许多误解的发生。随着系统的改进，字典中的信息也会发生变化，新的数据会随时加入进来。

数据字典用于定义数据流图中各个图形元素的具体内容，为数据流图中出现的图形元素做出确切的解释。数据字典包含四类条目：数据流、数据存储、数据项和数据加工。这些条目按照一定的规则组织在一起，以构成数据字典。在定义这些规则时，常用的符号如表 4-1 所示。

表 4-1 数据字典的常用符号

符号	含义	示例
=	被定义为	
+	与	X=a + b 表示 X 由 a 和 b 组成
[⋯|⋯]	或	X=[a | b]表示 X 由 a 或 b 组成
m{⋯}n	重复	X= 2{a}6 表示重复 2～6 次 a
{⋯}	重复	X={a}表示 X 由 0 个或多个 a 组成
(⋯)	可选	X=(a)表示 a 在 X 中可能出现，也可能不出现
"⋯"	基本数据元素	X="a"表示 X 是取值为字符 a 的数据元素
..	连接符	X=1..9 表示 X 可取 1～9 中的任意一个值

例如，数据流"应聘者名单"由若干个应聘者姓名、性别、年龄、专业、联系电话等数据项组成，那么"应聘者名单"可以表示为：应聘者名单={应聘者姓名+性别+年龄+专业+联系电话}。而数据项"考试成绩"可以表示为：考试成绩= 0..100。

又如，某教务系统的学生成绩数据库文件中的数据字典的描述可以表示为：

- 文件名：学生成绩
- 记录定义：学生成绩= 学号+姓名+{课程代码+成绩+[必修|选修]}
- 学号：由 6 位数字组成
- 姓名：2～4 个汉字
- 课程代码：8 位字符串
- 成绩：1～3 位十进制整数
- 文件组织：以学号为关键字递增排列

数据字典的实现方法既可以采用全人工过程，也可以采用全自动过程或者混合过程。无论使用哪一种方法来实现，数据字典都具有以下特点：

- 通过名字可以方便地查询数据定义。
- 能够容易地修改和更新信息。
- 不重复在规格说明中其他组成部分已经出现的信息。
- 可以单独处理描述每个数据元素的信息。
- 定义的书写方法简单并且严格。

4.4.3 结构化语言

结构化语言是一种介于自然语言和形式化语言之间的半形式化语言。虽然使用自然语言来描述加工逻辑是最为简单的，但是自然语言往往不够精确，可能存在二义性，而且很难用计算机处理。形式化语言可以非常精确地描述事物，而且还可以使用计算机来处理，但是往往用户却不容易理解。因此，可以采用一种结构化语言来描述加工逻辑，它是在自然语言的基础上加入了一定的限制，通过使用有限的词汇和有限的语句来严格地描述加工逻辑。

结构化语言主要使用的词汇包括：祈使句中的动词、数据字典中定义的名词或数据流图中定义过的名词或动词、基本控制结构中的关键词、自然语言中具有明确意义的动词和少量的自定义词汇等。一般不使用形容词或副词。另外，还可以使用一些简单的算术运算符、

逻辑运算符和关系运算符。

结构化语言中的三种基本结构的描述方法是：

（1）顺序结构，由自然语言中的简单祈使语句序列构成。

（2）选择结构，通常采用 IF … THEN … ELSE … ENDIF 结构和 CASE … OF … ENDCASE 结构。

（3）循环结构，通常采用 DO WHILE…ENDDO 结构和 REPEAT…UNTIL 结构。

【例 4-1】 某学院依据每个学生每学期已修课程的成绩制定奖励制度。如果优秀比例占 60％以上，并且表现优秀的学生可以获得一等奖学金，表现一般的学生可以获得二等奖学金；如果优秀比例占 40％以上，并且表现优秀的学生可以获得二等奖学金，表现一般的可以获得三等奖学金。

可以对上述例题用结构化语言描述加工逻辑，具体表现形式如下。

计算某一学生所获奖学金的等级：

```
IF 成绩优秀比例≥60% THEN
  IF 表现 = 优秀 THEN
    获得一等奖学金
  ELSE
    获得二等奖学金
  ENDIF
ELSEIF 成绩优秀比例≥40% THEN
  IF 表现 = 优秀 THEN
    获得二等奖学金
  ELSE
    获得三等奖学金
  ENDIF
ENDIF
```

4.4.4　判定表

判定表用来描述一些不容易用语言表达清楚或者需要很大篇幅才能用语言表达清楚的加工逻辑。在一些数据处理中，其数据流图的处理需要依赖于多个逻辑条件的取值，但是这些取值的组合可能构成多种不同的情况，对应地需要执行不同的动作，在这种情况下使用结构化语言来描述就显得很不方便，应该使用一种描述机制来清晰地表示复杂的条件组合与动作之间的对应关系。判定表就是解决这一问题的有力工具。

一张判定表由四个部分组成，如表 4-2 所示。

表 4-2　判定表的一般结构

所有的判断条件	各种条件的组合
所有的可能操作	条件组合对应的操作

判定表左上部列出所有的判断条件；左下部列出所有的可能操作；右上部的每一列表示各种条件的一种可能组合，填入 T 或 Y 表示条件成立，填入 F 或 N 表示条件不成立，空白表示条件成立与否都不影响操作；右下部的每一列表示一种条件组合相对应的

操作，填入“√”表示在该列上部规定的条件下做该行左边列出的操作，空白表示不做该项工作。

【例 4-2】 将例 4-1 中给出的奖励条件再进行细化。每个学生每学期已修课程成绩的比例情况：优秀比例占 60%以上、并且良以下比例小于 20%，而且表现优秀的学生可以获得一等奖学金，表现一般的学生可以获得二等奖学金；优秀比例占 60%以上、若良以下比例小于 30%，而且表现优秀的学生可以获得二等奖学金，表现一般的学生可以获得三等奖学金；若优秀比例占 40%以上、并且良以下比例小于 20%，而且表现优秀的学生可以获得二等奖学金，表现一般的学生可以获得三等奖学金。若良以下比例小于 30%，而且表现优秀的学生可以获得三等奖学金，表现一般的学生可以获得四等奖学金。

采用判定表给出加工逻辑，其主要步骤如下所示：

(1) 列出所有的判断条件，填写判定表的左上限。

(2) 列出所有的可能操作，填写判定表的左下限。

(3) 计算所有可能的、且有意义的条件组合，确定组合的个数，填写判定表的右上限。

(4) 将每种组合所指定的操作，填写在判定表右下限相应的位置。

(5) 合并相同的操作，简化规则。

(6) 将简化后的判定表重新排列。

奖学金发放判定表如表 4-3 所示。

表 4-3 奖学金发放判定表

条件	优秀比例≥60%	T	T	T	T	F	F	F	F
	优秀比例≥40%					T	T	T	T
	良以下比例≤20%	T	T	F	F	T	T	F	F
	良以下比例≤30%			T	T			T	T
	表现＝优秀	T	F	T	F	T	F	T	F
	表现＝一般	F	T	F	T	F	T	F	T
操作	一等奖学金	√							
	二等奖学金		√	√		√			
	三等奖学金				√		√	√	
	四等奖学金								√

4.4.5 判定树

判定树是判定表的图形化表示。由于判定表不直观，需要认真推敲才能看出其中的含义。判定树比判定表直观得多，它是采用一种树图方式来表示多种条件、多个取值所采取的操作。判定树的分支表示各种不同的条件，随着分支层次结构的扩充，各个条件完成自身的取值。判定树的叶子给出应完成的操作。

很明显，使用判定树的优点是直观、易于掌握和易于使用，但是它也有明显的缺点。判定树的简洁性不如判定表，数据元素的同一个值往往需要重复多遍，而且越接近判定树的叶子重复次数就越多。

【例 4-3】 采用判定树表示例 4-2，如图 4-9 所示。

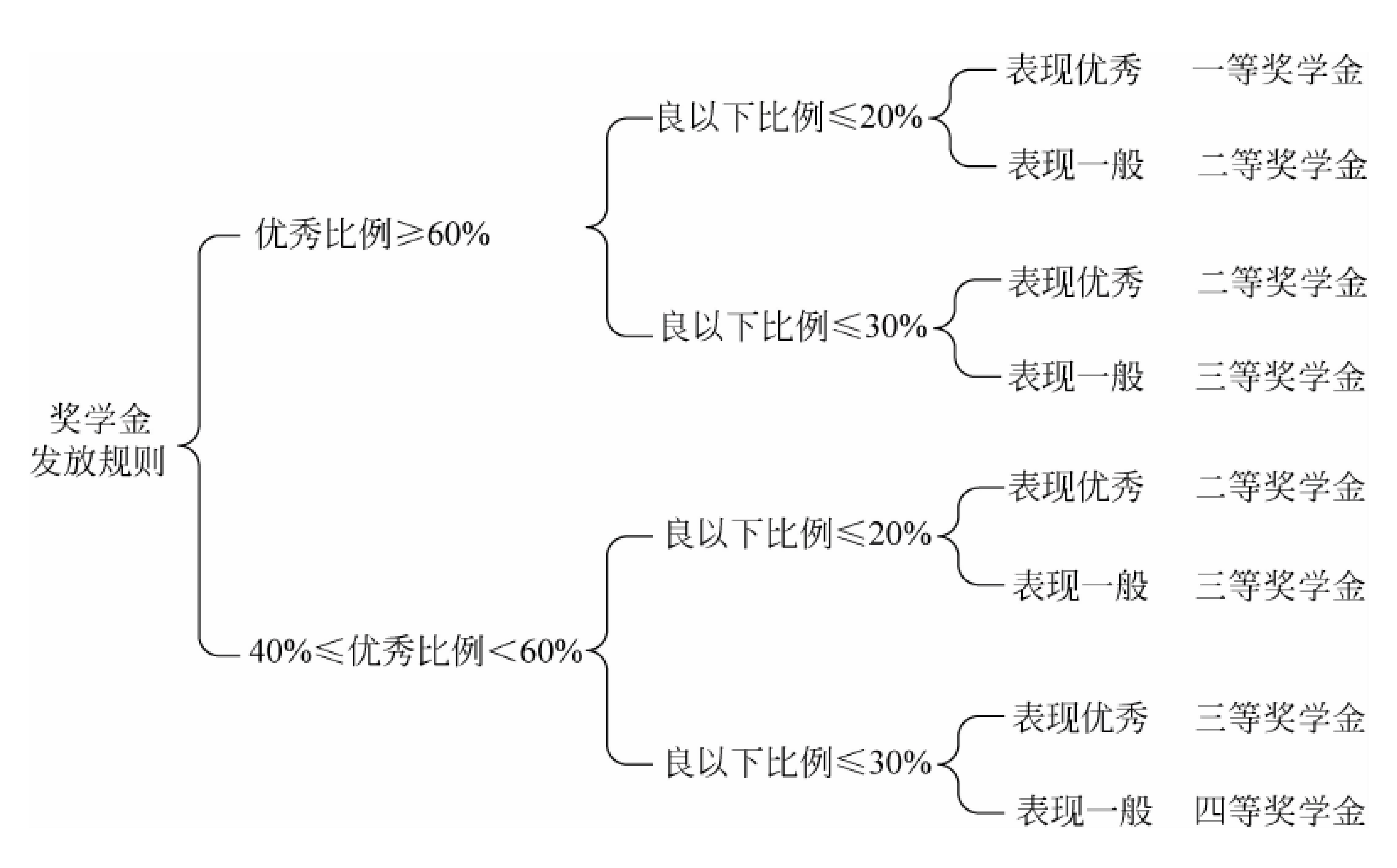


图 4-9 奖学金发放判定树

4.5 需求过程管理

需求过程管理的中心任务是保证软件项目满足用户在软件系统功能、性能和接口三个方面的需求。

4.5.1 需求分析阶段的项目管理

需求分析阶段是用户与系统开发人员协作最为密切的阶段之一，也是项目管理中难度最大的阶段，需要通过双方人员的共同努力才能取得成功。在此阶段，首先要做好需求分析的进度管理与质量管理工作，以确保需求分析工作按照预期进度、高质量地完成。

1. 对需求分析的进度管理

1）详细地制定工作计划

由于在需求分析阶段，项目的双方人员在工作上接触比较多，很容易出现配合上的矛盾和问题。所以，在需求分析阶段开始时，项目经理要与用户方的负责人员进行沟通，制定出本阶段的详细工作计划。计划主要应包括：本阶段详细的进度计划；项目双方参与人员的工作分工及职责要求；项目双方人员的工作时间约定、工作内容要求；项目协作过程中双方工作人员的工作流程；项目过程中可能出现的问题、解决方法的流程等。在计划完成后，要形成正式的书面文件，经项目双方负责人员签字认可，并且下发给项目组的所有成员。通过这种方法保证项目双方进行有效地工作配合。

2）科学地进行需求调研

在需求分析阶段，需要调研的需求因素很多，必须采用针对性的需求调研方法。在需求调研的一般工作流程中，第一件工作是需求调查准备工作。首先要确定需求调查的内容，其次应当确定需求调查的方式，最后需要确定调查的时间、地点、人员等。第二件工作是进行调查并记录。调查准备工作完毕后，需求分析人员将按照需求调查计划开展调查工作，在调

查过程中随时记录用户提出的需求信息。第三件工作是撰写用户需求说明书。需求调研工作结束后，需求分析人员需要对收集到的所有需求信息进行分析整理，去掉其中的错误信息，归纳并总结出共性的用户需求，完成用户需求说明书的编写工作。第四件工作是进行需求确认工作。

3）有效地遏制需求变更

软件的需求变更是软件项目开发和实施中的最大敌人，它可能发生在软件项目的各个阶段。所以对软件需求的变更控制应当贯穿于软件项目实施的各个阶段。在需求分析阶段，用户需求的变更主要表现为用户需求的反复，从而导致需求分析工作无法按计划完成。要遏制分析阶段的需求变更，一般可以采用的方法为进行详细周密地需求调研；用户签字制度的建立；定期的工作通报制度；将签字认可后的需求纳入需求管理等。在需求分析过程中，对所有需求分析项目进行分类管理，可以最大程度地降低需求变更的发生，将变更造成的影响降到最低。

4）广泛地建立用户关系

在需求分析阶段，分析人员要广泛与用户建立良好的工作关系，与用户的沟通要有一定的深度。在需求调查过程中，要进行全面、广泛地调研，要面向用户方的技术人员、主管人员以及参与的全体工作人员，详细了解用户方的整体需求细节，从不同角度掌握用户方的需求想法。除了与用户建立良好的工作关系外，还要努力建立深厚的私人关系，只有这样才能清楚地了解用户的真实想法，获得用户的支持。对于用户方的不同认识，分析人员可以通过召开项目协调会的方法，协调并统一用户方人员对相关需求的一致看法。

5）合理地完成需求验收

需求分析是经过不断反复的需求定义、文档记录、需求演进的过程，并最终在需求验收的基础上完成本阶段工作。要做好需求验收工作，需要踏踏实实地做好需求分析的各阶段工作：通过项目的合同条款，做好项目的范围规划，明确项目的工作内容；通过项目的工作计划，明确工作进度、人员分工及工作职责；做好各部分需求条款的签字验收工作及定期编写工作总结与工作汇报；做好目标系统的介绍或原型系统的演示。这样，需求分析工作可以确保按进度、高质量地完成，需求阶段的验收工作也可以顺利地进行。

需求分析的进度管理的内容如图 4-10 所示。

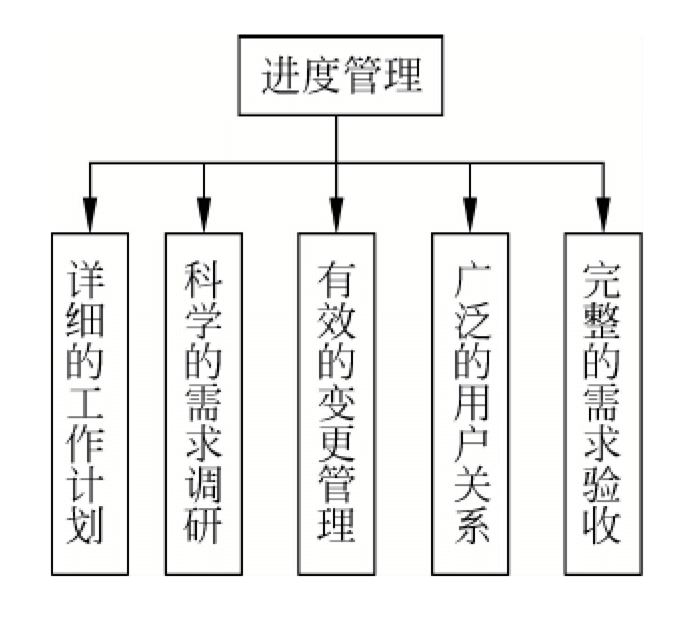


图 4-10　对需求分析的进度管理

2. 对需求分析的质量管理

在需求分析阶段，高质量的软件需求可以真实地反映用户的实际要求，可以减少整个项目中的变更，还可以大大地提高软件开发效率。对需求分析的质量管理应当做好下列工作。

1）认真进行调研准备

在需求分析阶段，分析人员在进行每一次需求调研之前，都要认真做好准备工作，按照工作计划设定需求调研主题；设计好所采用的调研方式与方法；估计可能出现的结果形式以及应对措施等。只有进行了认真、完善地准备工作，才能取得比较满意的需求调研结果。

2）正确理解用户需求

在需求调研过程中，对于用户所描述的软件需求，分析人员要正确理解，并且通过用户签字形式的需求验收和原型系统的演示，以保证用户的描述能够被正确理解，使项目双方人员之间达成共识。分析人员在进行记录或书写需求说明书时，要准确表达，避免出现二义性的描述。

3）有效确保用户签字

在需求分析阶段，具有用户评审及验收签字后的需求文档是最终软件系统能否通过验收的关键点之一。因此项目管理人员必须有效地利用需求管理的方法，将需求分析阶段的所有需求调研以及会议讨论的结果形成正式的书面文件，然后再经过用户审核签字进行妥善保存。

4）积极做好管理工作

在需求分析阶段，通过需求管理工作，一方面可以完成需求文档的版本控制及需求变更的控制工作，以防止频繁的修改所造成的需求内容混乱；另一方面通过有效的需求管理，可以大大提高软件需求文档的复用率。

5）定期开展会议交流

分析人员要想获得高质量的需求分析结果，需要定期召开用户方项目交流会议，将已取得的结果通报给全体用户，并对需求分析中出现的问题公开，供全体人员进行讨论，最终形成一致的意见，同时还要对已经完成的需求结果进行用户确认。

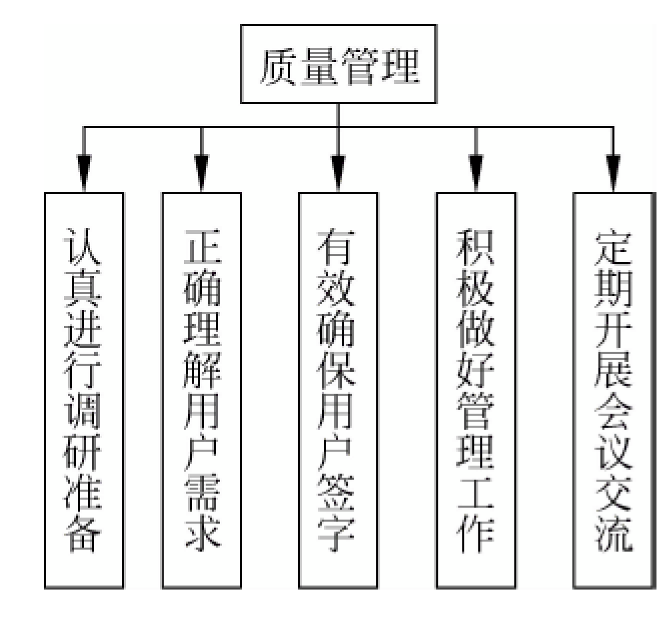


图 4-11　对需求分析的质量管理

需求分析质量管理的主要内容如图 4-11 所示。

4.5.2　需求过程管理的内容

需求过程管理的目标是在用户与开发人员对需求共同理解的基础上，维护软件计划、产品和活动与需求的一致性，并且保证需求在软件项目中得到实现。需求过程管理是面向需求过程的，主要包括需求确认、需求评审、需求跟踪和需求变更控制四个方面。

1. 需求确认

软件项目的开发是在需求确认的基础上进行的。需求确认是指开发方和用户方共同对需求文档进行评审，双方对需求达成共识后做出书面承诺，使需求文档具有商业合同效果。

2. 需求评审

为了保证软件需求定义的质量，在需求确认之后，需要软件项目评审组进行需求评审。评审的内容主要包括相关需求文档的评审。

3. 需求跟踪

需求跟踪是指通过比较需求文档与后续工作成果之间的对应关系，确保软件产品依据需求文档进行开发。

4. 需求变更控制

需求变更控制是指依据“变更申请-审批-更改-重新确认”的流程处理需求的变更，以防止需求变更失去控制从而导致项目不能顺利完成。

下面是一份需求变更控制报告。

需求变更控制报告

需求变更申请	
申请变更的需求文档	输入名称、版本、日期等信息
变更的内容及其理由	
评估需求变更将对项目造成的影响	
申请人签字	
变更申请的审批意见	
项目经理签字	审批意见： 签字： 日期：
用户签字	审批意见： 签字： 日期：
更改需求文档	
变更后的需求文档	输入名称、版本、完成日期等信息
更改人签字	
重新评审需求文档	
需求评审小组签字	评审意见： 签字： 日期：
变更结束	
项目经理签字	签字： 日期：

4.6 需求分析文档

在需求分析阶段，已经确定的用户需求应当得到清晰而准确地描述，软件需求规格说明书是描述需求的主要文档，它以完整的、正确的方式，表达了目标系统应该实现的用户需求。软件需求规格说明书应围绕以下四个方面组织。

（1）系统规格方面。这方面主要包括目标系统的总体概貌；系统功能、性能的要求；系统运行的要求；将来可能的修改扩充要求。

（2）数据要求方面。这方面主要包括建立数据字典，描绘系统数据要求，给出系统逻辑模型的准确的、完整的定义。

(3) 用户描述方面。从用户使用角度对系统进行描述，主要包括系统功能、性能概述，预期的系统使用步骤与方法、用户运行维护要求等。

(4) 开发计划方面。经过需求分析，对系统开发的成本估计、资源使用要求、项目进度计划等的要求。

4.6.1　需求文档完成的目标

软件需求规格说明书是软件工程项目的重要文档，它相当于用户和开发者之间的一项合同。软件需求规格说明书清楚地描述了软件产品做什么，以及产品的约束条件。它为软件设计提供了一个蓝图，为系统验收提供了一个标准集。所以软件需求规格说明书应当完成下列目标：

(1) 在软件产品完成方面，软件需求规格说明书为用户和软件设计人员之间建立的共同协议创立了一个基础，对要实现的软件功能做出了一个全面描述，帮助用户判断所开发的软件系统是否符合他们的要求，或者如何修改软件才能适合他们的要求。

(2) 在提高系统的开发效率方面，编制软件需求规格说明书的过程，可以使用户在开始设计之前周密地思考其全部需求，从而减少以后可能的重新设计、重新编码和重新测试所带来的返工。在软件需求规格说明书中，对各种需求认真地进行复查，还可以在开发早期发现若干遗漏、错误和不一致，以便及时加以纠正。

(3) 在成本计价和编制计划进度方面，软件需求规格说明书对所开发的软件系统的描述，是软件系统成本核算的基础，并且可以为各方面的费用提供依据。软件需求规格说明书对软件产品的清晰描述，有助于估计所有可能用到的资源，并且可以作为编制计划进度的依据。

(4) 在软件系统的移植性方面，有了软件需求规格说明书，可以容易地开发出可移植的软件系统，从而适应新的用户或新的应用平台。用户也易于移植其软件系统到其他部门，而软件设计人员同样也易于把系统移植给新的用户。

(5) 软件需求规格说明书是软件系统不断提高的基础。由于软件需求规格说明书所讨论的是软件系统，而不是开发这个系统的设计，因此软件需求规格说明书是软件系统继续提高的基础。虽然软件需求说明书也可能会被修改，但是原来的软件需求规格说明书还是软件系统进行改进的可靠基础。

4.6.2　需求文档的特点

一份好的软件需求规格说明书应该具有以下特点。

(1) 正确性：是指软件需求规格说明书应当正确地反映用户的真实意图，正确性是需求规格说明书最重要的属性。为了确保各项需求是正确的，开发人员和用户必须对软件需求规格说明书进行确认。

(2) 完整性：是指软件需求规格说明书中没有遗漏一些必要的需求。应该包括该系统包含的全部重要的用户需求；规定每种输入数据的软件响应；全部的术语、图表及文档必须完整，符合需求规范标准。

(3) 一致性：软件需求规格说明书中的各项功能、性能要求应该是相容的，不能互相发

生冲突。如描述同一对象不能存在两个以上的不同术语；要求的某一数据的内部属性不能产生矛盾；两个规定的处理在时间上不能产生矛盾。

(4) 清晰性：清晰的需求让人易读易懂，可以采用反问的方式判断软件需求规格说明书是否清晰：

- 文档的结构、段落是否混乱？上下文是否不连贯？
- 文档的语句是否含糊其辞？
- 文档的内容是否表达不明确？

(5) 可验证性：是指软件需求规格说明书中的每个功能、性能需求存在有限的人工或机器执行的过程，以确认该需求是否符合用户要求。如"软件系统具有良好的用户界面"的要求，用户和开发人员可以有着不相同的理解，因此是不可验证的。

(6) 可修改性：是指软件需求规格说明书的组织结构在需求发生变化时，对需求的修改能够保证其完整和一致。如果存在需求规格说明内容的列表、索引和交叉引用表，则当某个需求发生变化时，就可以方便地对软件需求规格说明书中必须修改的部分进行定位和修改。

(7) 可跟踪性：是指在软件系统开发中，每个需求在软件需求规格说明书中可以追溯出其来源。实现可跟踪性的常用方法是对软件需求规格说明书中的每个段落按层编号，每个需求给予唯一编码，使用特殊指示字对同一需求在不同出现地方进行标识。

4.6.3 需求文档编写的一般原则

对于用户需求通常有三种方法编写其需求规格说明书。第一个方法是用户自己描述并且自己编写需求；第二个方法是以用户为主、开发人员和用户共同编写需求；第三个方法是开发人员代替用户编写需求。因为用户是最终使用软件系统的人，对需求有着比较深入的了解，但是用户缺乏编写需求的技巧和方法，需要开发人员的指导，因此第二种方法比较好。在编写软件需求规格说明书的过程中要遵循以下基本原则。

1. 用户观点

一份好的用户需求应该是站在用户的角度来思考问题，是用户能够利用系统来完成什么，而不是系统自己能够完成什么。

2. 整体观念

在软件系统开发初期，最关键的是建立一个高层的需求概况，而不是立即深入到细节。因此需要尽可能全面地发现需求，以及维持一个简单的需求列表。

3. 评估依据

以用户为主编写的需求为软件系统的评估提供了依据，在需求初期就进行适当地估算，可以让用户有一个直观的成本概念，为用户在制定需求实现的先后次序上提供了指导。

4. 统筹安排

在制定最终需求时，虽然用户需求都是有用的，但在次序和数量上是不同的。在每一个

用户需求具有了成本之后,用户就能够权衡实际成本和需求,安排需求的位置。

用户对最终需求的选择直接影响到下一步的计划制定。在第一个版本中,用户希望能够实现哪些需求,经过估算后,这些需求是否还能够在这个版本中实现,且需要多长的时间等,这些都是需求对计划的影响。

4.6.4 需求文档编写格式

编写需求文档(软件需求规格说明书)是为了使用户和开发人员对该软件系统的初始规定有一个共同的理解,使之成为整个开发工作的基础。每个软件开发组织都应该在其开发的项目中采用一种标准的软件需求规格说明模板来编写软件需求规格说明书。目前已有多种实用的模板可以使用,其中来自 IEEE 标准 830-1998 的模板——"IEEE 推荐的软件需求规格说明的方法"(IEEE 1998)——就是一个结构良好、适用多种软件项目的、灵活的模板。下面就以一个从 IEEE 830 标准改写并扩充的软件需求规格说明书模板为例介绍软件需求规格说明书的编写工作。

软件需求规格说明书模板

1. 引言

引言提出了对软件需求规格说明的纵览,有助于读者理解文档是如何编写的,以及如何阅读和解释文档。

1.1 编写目的

说明编写这份软件需求规格说明书的目的,并指出预期的读者。

1.2 背景说明

- 拟开发的软件系统的名称。
- 本项目的任务提出者、开发者、用户以及实现该软件系统的计算中心或计算机网络。
- 该软件系统同其他系统或其他机构的基本相互来往关系。

1.3 定义

列出本文档中用到的专门术语的定义和外文首字母缩写词的原词组。

1.4 参考资料

列出编写软件需求规格说明书时所需要的参考资料,如:

- 本项目经核准的计划任务书或合同、上级的批文。
- 属于本项目的其他已发表的文件。
- 本文档中各处引用的文件、资料以及所要用到的软件开发标准。列出这些文件资料的标题、文件编号、发表日期和出版单位。

2. 任务概述

2.1 目标

叙述该项软件系统开发的意图、应用目标、作用范围以及其他应向用户说明的有关该软件开发的背景材料。解释被开发软件与其他有关软件之间的关系。如果本软件产品是一项独立的软件,而且全部内容自含,则应明确说明。如果所定义的产品是一个更大系统的组成部分,则应当说明本产品与该系统中其他各组成部分之间的关系,为此可以使用一张方框图来说明该系统的组成和本产品与其他各部分的联系和接口。

2.2 用户特点

列出本软件系统的最终用户的特点，充分说明操作人员、维护人员的教育水平和技术专长，以及本软件系统的预期使用频率。

2.3 假定与约束

列出进行本软件系统开发工作的假定和约束条件，例如经费限制、开发期限等。

3. 需求规定

3.1 对功能的规定

用列表的方式逐项定量和定性地叙述对软件系统所提出的功能要求，说明输入什么值、经过怎样的处理、可以得到什么输出，并说明软件系统应支持的终端数目和应支持的并行操作的用户数目。

3.2 对性能的规定

3.2.1 精度

说明对该软件系统的输入、输出数据的精度要求。

3.2.2 时间特性要求

说明对于该软件系统的时间特性要求，如：

- 响应时间。
- 更新处理时间。
- 数据转换和传送时间。

3.2.3 灵活性

说明对该软件系统的灵活性要求，即当需求发生变化时，该软件系统对这些变化的适应能力，如：

- 操作方式的变化。
- 运行环境的变化。
- 同其他软件接口的变化。
- 精度和有效时限的变化。
- 计划的变化。

对于为了提供这些灵活性而进行的专门设计部分，应该加以标明。

3.3 输入输出要求

解释各输入输出的数据类型，并逐项说明其媒体、格式、数值范围、精度等。对软件系统的数据输出及必须标明的控制输出量进行解释。

3.4 数据管理能力要求

说明需要管理的数据表和记录的大小规模，要按可预见的增长对数据及其分量的存储要求做出估算。

3.5 故障处理要求

列出可能的软件、硬件故障以及对各项性能所产生的后果和对故障处理的要求。

3.6 其他专门要求

如用户对安全保密的要求，对使用方便性的要求，对可维护性、可补充性、易读性、可靠性、运行环境可转换性的特殊要求等。

4. 运行环境规定

4.1　设备

列出运行该软件系统所需要的硬件设备，主要包括：

- 处理器型号及内存容量。
- 外存容量、联机或脱机、媒体及其存储格式，设备的型号及数量。
- 输入和输出设备的型号和数量，联机或脱机。
- 数据通信设备的型号和数量。
- 功能键及其他专用硬件。

4.2　支持软件

列出支持软件，包括要用到的操作系统、编译(或汇编)程序、测试软件等。

4.3　接口

说明该软件系统同其他软件之间的接口、数据通信协议等。

4.4　控制

说明控制该软件系统的运行方法和控制信号，并说明这些控制信号的来源。

5. 需求分析

附录

索引

需要指出的是，软件需求的内容，是反映用户对拟开发系统特性的要求，而不是反映系统的开发特性。因此在软件需求规格说明书中一般不包括软件设计、开发里程碑、开发详细费用等方面的内容。

4.7　需求评审

有时由分析人员所提供的软件需求规格说明书初看起来是正确的，但是在具体实现时就有可能出现一些问题，如需求不清楚、需求不一致等。有时以需求说明书为依据编写测试计划时，也会发现需求说明书中存在有二义性。为了对需求分析阶段的工作进行验证和完善，应该对软件功能的正确性、软件需求说明书的一致性、完整性和准确性以及其他需求予以评审。

4.7.1　需求评审的方法

需求评审具体所做的工作可以归纳为以下四个方面。

1. 审查需求文档

对于所提交的需求文档，必须进行全面、认真地审查，以防止理解错误和遗漏需求情况的发生。组织一个包括分析人员、用户、设计人员、测试人员等不同代表组成的小组，对需求规格说明书以及相关模型进行认真的检查，对于保证软件质量而言是一个非常有效的方法。

2. 设计测试用例

通过阅读需求规格说明书，一般很难确定在特定环境下系统的行为，可以将功能需求作

为基础设计测试用例，让用户通过使用测试用例来确认是否达到了期望的要求。还可以从测试用例追溯回功能需求以确保没有需求会被遗漏，并且确保所有测试结果与测试用例相一致。同时要使用测试用例来验证需求模型的正确性。

3. 编写用户手册

在需求开发早期起草的用户手册，可以作为需求规格说明书的参考资料来辅助需求分析，因为一份好的用户手册一般是用浅显易懂的语言描述出所有对用户可见的功能。

4. 确定合格标准

在需求评审阶段，需要确定合格的评审标准，让用户描述什么样的软件系统才是他们需要的和适合他们使用的。应将合格的测试建立在使用情景描述或使用实例的基础之上。

需求评审究竟需要评审什么？通常要细致到什么程度？严格地讲，应当检查需求文档中的每一个需求、每一行文字、每一张图表。评审需求优劣的主要指标有，正确性、清晰性、一致性、必要性、完备性、可实现性、可验证性等。

为了保证软件需求定义的质量，通常评审工作应当由专门指定的人员负责，并按照规定程序严格进行。用户、开发部门的管理者、软件设计人员、软件实现人员、软件测试人员都应当参加评审工作。在评审结束时应该有负责人的结论意见及签字，然后就可以转入设计阶段。

4.7.2 需求评审的内容

需求文档的评审是一项精益求精的技术，它可以发现那些具有二义性的或不确定的需求以及那些由于定义不清而不能作为设计基础的需求。评审的主要内容有：

- 系统定义的目标是否与用户的要求相一致。
- 系统需求分析阶段所提供的文档资料是否齐全。
- 需求文档中的描述是否完整、清晰、准确地反映了用户要求。
- 与所有其他系统成分的重要接口是否都已经被描述。
- 所开发项目的数据流与数据结构是否充足且确定。
- 所有图表是否清楚，在不补充说明时能否被理解。
- 系统主要功能是否已经包括在规定的软件范围之内。
- 设计的约束条件或限制条件是否符合实际要求。
- 开发过程中的技术风险有哪些。
- 是否考虑过将来可能提出的软件需求。
- 是否详细制定了检验标准，它们能否对系统定义的成败进行确认。
- 用户是否检查了初步的用户手册。
- 软件开发计划中的估算是否受到了影响。
- 软件需求中是否还有遗漏、重复或不一致的方面。

需求评审一般按照预先定义好的步骤进行，评审内容需要记录在案，包括确定材料，评审员、评审小组对产品是否完整或是否需要开展进一步工作的评判，以及对所发现的错误和

所提出问题的总结。评审小组成员对于评审的质量负责，而开发者对于所开发产品的质量负责。

4.7.3　需求评审的测试

需求评审的测试是对软件的需求分析、需求规格说明的最终审查，是质量保证工作中最为关键的一个环节。与测试相近的一个名词是纠错，其目的是定位和纠正错误，保证软件的质量。测试过程如图 4-12 所示。

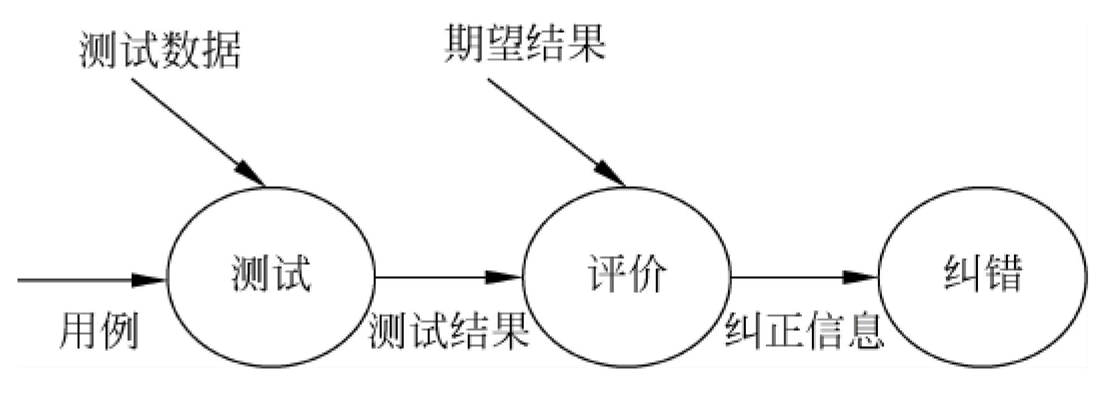


图 4-12　测试过程

如果只是通过阅读软件需求规格说明书，是很难想象出系统在特定环境下的行为的。以功能需求为基础的测试用例可以使项目的参与者了解系统的行为。虽然不可能在实际系统上执行测试用例，但是设计测试用例的过程就可以解释许多需求问题。如果在部分需求稳定时就开始设计测试用例，就可以及早发现问题并以较少的费用解决这些问题。

常用的测试方法有黑盒测试和白盒测试。

黑盒测试是把系统看成是一个黑盒子，人们完全不用考虑程序的内部结构和处理过程。黑盒测试是在系统接口进行的测试，它只检查系统功能是否按照规格说明书的规定正常使用，系统是否能够正常接收输入数据以产生正确的输出信息，并且保持外部信息的完整性。使用黑盒测试法设计测试用例，有三种常用技术：等价类划分法、边界值分析法和错误猜测法。黑盒测试又被称为功能测试。

白盒测试是把系统看成一个透明的白盒子，人们可以完全了解系统的结构和处理过程。这种方法按照系统内部的逻辑测试程序，检验系统中的通路是否都能按照预定要求正确工作。使用白盒测试法设计测试用例，有语句覆盖、分支覆盖、条件覆盖、路径覆盖等方法，主要目的是提高测试的覆盖率。白盒测试又称结构测试。

在开发过程的早期阶段，可以从使用实例中获得概念上的功能测试用例，然后就可以利用测试用例来验证需求规格说明书和分析模型。当分析人员、开发人员和用户通过测试用例进行研究时，他们将对系统如何运行的问题有着更为清晰的认识。这些基于模拟使用的测试用例可以作为用户验收测试的基础。在正式的系统测试中，可以把它们细化为测试用例和过程。

4.8　本章小结

软件项目的需求分析是一个软件项目的开端，也是项目建设的基石。本章主要介绍了软件需求分析的基本概念、需求分析的任务、需求分析的方法、需求的描述工具、需求的过程

管理、需求分析文档的编写以及需求的评审过程。

在软件工程中，软件需求分析是指在建立一个新的软件系统或者改变一个现存的软件系统时，描述新系统的目的、范围、定义和功能时所做的全部工作，主要包括获取用户需求、分析用户需求、编写需求文档、通过需求评审等几个过程。

软件需求分析的任务就是深入描述软件的功能和性能，确定软件设计的限制和软件同其他系统的接口细节，定义软件的其他有效性需求，即对目标系统实现的功能提出完整的、准确的、清晰的、具体的要求。

软件需求分析的方法是由对软件问题的信息域和功能域的系统分析过程及其表示方法所组成。需求分析的方法有很多种，根据目标系统被分解的方式不同，可以分为结构化分析方法、面向对象分析方法和面向问题域分析方法三种方法。

结构化分析方法是把软件系统功能作为一个大的模块，根据分析与设计的不同要求，进行模块分解或者组合。而面向对象分析方法是一种运用对象、类、继承、封装、聚合、消息传送、多态性等概念来构造系统的一种软件分析方法，在分析中系统分析人员要深入理解用户需求，抽象出系统的本质属性，提取系统需求规格说明，并用模型准确地表示出来。面向问题域分析方法是一种比较新的技术，它更多地强调描述，较少强调建模。在面向问题域的分析中，问题框架将作为一个全新的模型被引用，该模型不仅有助于把需求从问题域的内在性质中划分出来，还有助于建立问题域的类型。

统一建模语言（UML）是一种标准的图形化建模语言，它主要用于软件的分析与设计，其主要特点就是使开发人员用标准的、容易理解的方式建立起充分表达设计思想的系统模型和结构。UML 适用于系统开发过程中从需求规格描述到系统完成后的测试的各个不同阶段。

软件需求分析的描述工具通常有数据流图、数据字典、判定表、判定树、结构化自然语言等。数据流图（DFD）是一种从数据传递和加工的角度、以图形的方式描述数据流从输入到输出的移动变换过程。数据字典是为数据流图中的每一个数据流、文件、加工以及组成数据流或文件的数据项做出说明。结构化语言是一种介于自然语言和形式化语言之间的半形式化语言，它是在自然语言的基础上加入了一定的限制，通过使用有限的词汇和有限的语句来严格地描述加工逻辑。判断表是用来描述一些不容易用语言表达清楚或者需要很大篇幅才能用语言表达清楚的加工逻辑，而判定树是判定表的图形化表示。

软件需求过程管理的中心任务是保证软件项目满足用户在软件系统功能、性能和接口三个方面的需求。在此阶段，应主要做好需求分析的进度管理和质量管理工作，以确保需求分析工作按照预期进度、高质量地完成。

软件需求规格说明书是描述软件需求的主要文档，它以完整的、正确的方式，表达了目标系统应该实现的用户需求。软件需求规格说明书围绕系统规格方面、数据要求方面、用户描述方面和开发计划方面等四个方面进行组织和编写。

为了对需求分析阶段的工作进行验证和完善，还必须对软件功能的正确性、软件需求规格说明书的一致性、完整性和准确性以及其他需求予以评审。需求评审具体所做的工作可以归纳为审查需求文档、设计测试用例、编写用户手册和确定合格标准四个方面的工作。

习题 4

1. 为什么要进行软件需求分析？请叙述软件需求分析的主要过程。
2. 什么是结构化分析？其结构化体现在哪里？
3. 什么是面向对象分析？其主要思想是什么？
4. 软件需求分析的原则主要有哪些？
5. 软件需求分析的描述工具有哪些？
6. 软件需求规格说明书由哪些部分组成？
7. 数据字典包括哪些内容？它的作用是什么？
8. 需求评审的主要内容是什么？
9. 怎样建立目标系统的逻辑模型？需要经过哪些步骤？
10. 需求变更的原因主要有哪些？如何应对需求变更？

第5章 软件策划

5.1 软件策划概论

软件策划又叫软件项目策划，既能为软件开发者和管理者制定合理的工作计划，又能为软件项目跟踪和监控提供考核依据。软件策划属于软件管理和软件决策的范畴，是项目经理以上人员的职责范围，是软件企业管理的重大事件之一。要使软件策划工作十分准确，往往十分困难。只有达到CMMI三级以上的软件组织，在其强大的软件测量数据库和软件工程数据库的支持下，其策划工作的误差才能控制在20%以内。到了CMMI四级，其策划工作的误差才能控制在10%以内。到了CMMI五级，其策划工作的误差才能控制在5%以内。

软件策划输入的文档是：

- 用户需求报告；
- 项目合同及其附件或立项建议书；
- 软件项目任务书。

在软件策划过程中，结合公司的过程数据库，输出的文档是：

- 工作任务分解表；
- 风险评估表；
- 软件项目风险管理计划；
- 软件项目开发计划（包括质量保证计划、配置管理计划、软件测试计划、里程碑及评审计划）。

1. 软件策划的目的

软件策划的目的是为软件开发和软件管理制定合理的计划。由于项目的管理者按照计划确定的内容和进度对项目进行管理，所以计划是否合理直接关系到项目管理的成败。通常情况下，在项目启动时，就要做好项目策划，并随着工程的进展不断地加以精化。

2. 软件策划的基础

软件策划的基础就是进行软件生命周期模型的选择。软件开发组织和项目经理要根据项目的特点，在瀑布模型、增量模型、迭代模型、原型模型等生命周期模型中选择一种，经过适当的裁剪后，列入项目计划，作为软件项目策划的理论基础之一。

3. 软件策划的目标

软件策划是项目经理和高层经理管理项目的依据，是项目跟踪和监控的基础。软件策划要实现的具体目标有三个。

（1）第一个目标是建立三个估计文档。

- 软件工作产品规模和工作量估计
- 软件成本估计
- 计算机资源估计

这三个估计文档是供项目策划和跟踪使用的。

（2）第二个目标是对软件项目的各种活动和约定建立计划文档。活动包括开发活动和管理活动；约定是指对项目的各种标准、规范、规程的约束。

（3）第三个目标是针对受影响的组和个人，同意他们对软件项目进行约定。受影响的组和个人有：

- 软件工程组（项目组）；
- 软件估计组；
- 系统测试组；
- 质量保证组；
- 配置管理组；
- 合同管理组；
- 文档支持组。

其中，有的组可能只有一个人。对于一些小型项目，也可能一个人身兼数职。

4. 软件策划的方法

软件策划也要讲究方法，要用科学管理的思想来进行策划。软件策划的方法一般是采用经验数据加结构化方法，包括三个要点。

（1）信息粒度（Granularity）由粗到细地分解：自顶向下、逐步细化、逐项逐条逐日安排计划。

（2）信息粒度由细到粗地综合：自底向上、逐步归纳、逐日逐周逐月安排计划。

（3）同类项目经验数据类比法、同行专家协商策划法。

粒度是指数据仓库的数据单位中保存数据的细化或综合程序的级别，粒度问题是设计数据仓库的一个最重要的方面。信息粒度是指所需信息的详细程度，高粒度的信息存储空间小，查询处理性能好，但是结构数据仓库的外围的转换概括等工作较复杂。越细化的数据粒度级越低，越综合的数据粒度级越高。

软件策划以用户确认的需求为基础，以软件机构内部的软件标准为依据，将机构内部类似项目的成功经验作为策划的参考依据。

5. 软件策划的时机

对软件项目进行策划的时机，我国习惯的作法与国际通用的作法不大一致。国际通用作法是先做需求分析，后做软件策划，因为需求不清楚，项目的功能点个数、性能点个数、接

口个数、界面个数、实体个数、文档页数都心中无数，策划人员是无法估计工作量、进度、经费和其他资源的，完成项目策划是不现实的。

与国际通用作法相反，我国习惯的作法是在用户需求报告之前策划，不习惯在用户需求报告之后策划。不管怎么样，调查研究是十月怀胎，软件策划是一朝分娩，心中无数是不能做软件策划的。因此，我们要逐渐去向国际接轨。

软件策划的最佳时机是：第一步，软件策划至少要在软件《合同》、《立项建议书》和《任务书》制定之后进行；第二步，软件策划要在《用户需求报告》制定之后，而在《需求规格说明书》制定之前进行。

5.2 软件策划过程

软件策划的目的是为实施软件工程和管理软件项目制定合理的计划。软件策划包括估计待完成的工作，确定进行该工作的计划。该计划为实施和管理软件项目活动提供基础，并根据软件项目的资源、约束条件和能力，向软件项目的客户提供承诺。软件策划过程包含以下几个主要阶段。

1. 定义软件过程

所谓定义软件过程，就是根据选择的软件生命周期模型，规定软件的开发阶段，以及每一阶段的工作步骤和文档标准等内容。在项目策划时期，先根据项目特性，使用软件生命周期模型对项目中将要进行的软件工程过程进行描述；再根据项目自身的特点，对项目的类型进行详细划分；然后根据软件组织制定的《生命周期模型裁剪指南》，基于已确定的软件生命周期模型，对标准软件过程进行裁剪，形成项目定义软件过程；最后使用项目定义软件过程，指导项目策划活动的进行。

2. 进行软件估计

所谓软件估计，就是指对软件项目进行量化估计，并记录估计结果的过程。软件估计是软件度量的一部分，它既是软件策划的核心，又是软件策划的重点与难点。

在软件项目中，项目组要对项目的规模、工作量、成本、进度、关键计算机资源等方面进行量化估计，然后使用估计数据进行软件策划。在以后的项目执行过程中，将不断收集到的项目实际数据与估计数据进行比较，从而了解项目的进展状态。若发现估计数据严重偏离实际数据，则要重新进行软件估计。这些收集的实际项目数据与估计数据，要被及时地录入到软件测量数据库之中，日积月累，就建立了强大的软件过程数据库，为日后的软件策划和CMMI 升级积累了丰富的基础数据。

3. 进行风险分析

所谓风险分析，是通过对项目的各个方面可能存在的风险进行识别和分析，逐步降低与化解风险，确定避免或减轻风险的策略及措施，以达到回避风险、保证项目顺利进行的目的。风险分析最后会制定用于跟踪和监控风险的风险管理计划。软件一般存在五种风险，如表 5-1 所示。

表 5-1 软件风险的种类

序 号	风险名称	风 险 内 容
1	政策风险	IT 企业外部和 IT 企业内部两个方面的政策及政策的变化，将会给项目带来什么风险
2	技术风险	新技术的成熟程度及难度系数，将会给项目带来什么风险
3	技能风险	项目组成员学习、领会、掌握、运用新技术的能力，将会给项目带来什么风险
4	资源风险	保证项目正常进行所需的各种资源的供应程度，将会给项目带来什么风险
5	其他风险	目前意想不到的风险，即不可预测的风险，如天灾人祸

4. 进行项目跟踪与监督

所谓软件项目跟踪与监督，就是对策划阶段输出的软件开发计划，进行动态跟踪与实时监督，一旦发现偏差，必须立即纠正。

在《软件开发计划》中，描述了如何实施和管理项目定义的软件过程。在项目实践中，通常为项目指定一名项目软件经理(Project Software Manager)，由项目软件经理负责，依据开发计划对项目实施跟踪与监督，并在项目的执行过程中，要求项目的各级负责人查阅和分析软件测量数据库和文档库，定期地或以事件驱动式地对开发计划进行修订。

5. 完成软件开发计划书

软件开发计划是用于指导组织、实施、协调和控制软件研发与建设的重要文件，主要使项目成员有明确的分工及工作目标，并对拟开发项目的费用、时间、进度、人员组织、硬件设备的配置、软件开发环境和运行环境的配置等进行说明和计划。

由于软件项目能共享过程数据，所以在制定计划时，能吸取软件组织中积累的经验教训。为此，要建立较完善的软件测量数据库和文档库(这一工作称做过程财富积累)，一般在 CMMI 2 级就要开始考虑，在 CMMI 3 级就必须做到。

软件开发计划采用自然语言描述，可以在描述中加入图表。编制工具可以采用 Microsoft 公司的 Word 和 Project，一般采用 Word 书写文件的主体部分，采用 Project 形成的文件作为它的附件。

6. 评审项目的各种计划

评审项目的各种计划，获得参与软件开发计划制定的组或个人的同意，并得到高级管理者的批准。表 5-2 列出了软件开发计划制定的组或个人。

表 5-2 软件开发计划制定的组或个人

序 号	组或个人名称	组或个人名称说明
1	软件工程组	软件工程组
2	系统工程组	系统工程组是负责下列工作的个人的集合：规定系统需求；将系统需求分配给硬件、软件和其他成分；规定硬件、软件和其他成分之间的界面；以及监控这些成分的设计和开发，以保证它们符合其规格说明(系统工程组就是总体设计组)

续表

序　号	组或个人名称	组或个人名称说明
3	质量保证组	质量保证组是一些计划和实施项目的质量保证活动的个人的集合，工作的目的是保证遵守软件过程的步骤和标准
4	高级管理者	高级管理者在高的层次上履行管理职责
5	项目经理	项目经理对整个项目的总体业务负责；项目经理是指导、控制、管理和调整项目完成构造软件或硬件系统工作的个人，项目经理是最终向顾客负责的个人

5.3 软件估计的方法

软件估计历来是比较复杂的，因为软件本身的复杂性、历史经验的缺乏、估算工具缺乏以及一些人为错误，导致软件项目的规模估算往往和实际情况相差甚远。因此，估算错误已被列入软件项目失败的原因之一。

1. 软件规模和工作量估计方法

对软件工作产品的规模和工作量进行量化估计，早期的估计方法如表 5-3 所示。

表 5-3　软件工作产品规模和工作量的估计方法

序号	规模估计方法	工作量估计方法	工作量估计方法说明
1	功能点个数	N 个功能点/人月	一个人的月工作量，能完成的功能点个数
2	性能点个数	N 个性能点/人月	一个人的月工作量，能完成的性能点个数
3	代码行数	N 行代码/人月	一个人的月工作量，能完成的代码行数
4	实体个数	N 个实体/人月	一个人的月工作量，能完成的实体个数
5	需求个数	N 个需求数/人月	一个人的月工作量，能完成的需求个数
6	文档页数	N 页文档/人月	一个人的月工作量，能完成的文档页数

早期的估计方法都是面向过程的，因为只有面向过程的语言，其源程序的规模与工作量，才能比较准确地用代码行(Line Of Code，LOC)来计算。LOC 指所有的可执行的源代码行数，包括数据定义、数据类型声明、等价声明、输入输出格式声明等。1 代码行(1 LOC)的价值和每人月代码行数可以体现一个软件生产组织的生产能力。组织可以根据对历史项目的审计来核算组织的单行代码价值。

例如，某软件公司统计发现该公司每一万行 C 语言源代码形成的源文件约为 250KB。某项目的源文件大小为 3.75MB，则可估计该项目源代码大约为十五万行，该项目累计投入工作量为 240 人月，每人月费用为 10 000 元(包括人均工资、福利、办公费用公摊等)，则该项目中 1LOC 的价值为：

$$(240 \times 10\,000)/150\,000 = 16(\text{元/LOC})$$

那么，项目的人月均代码行数为：

$$150\,000/240 = 625(\text{LOC/人月})$$

以上是早期用代码行方法对软件规模和工作量进行的估算。在面向对象和面向元数据盛行的今天，准确的代码行是很难估计的。下面介绍几种常用软件项目规模和工作量的估

计方法。

1）希腊古都法

希腊古都法又称 Delphi 法。在没有历史数据的情况下，希腊古都法是最流行的专家评估技术，这种方式适用于评定过去的技术与将来的新技术与特定技术之间的差别，但专家的专业程度及对项目的理解程度是工作的难点，尽管希腊古都法可以减少这种偏差，但在评定一个新软件实际成本时用得并不多。希腊古都法鼓励参加者就问题相互讨论，要求有多种软件相关经验人的参与，互相说服对方。希腊古都法要求若干专家参与，并且要选出一名组长或估计协调人，由他组织软件估计。使用希腊古都法的基本步骤如下。

（1）协调人向各专家提供项目规格和估计表格。

（2）协调人召集小组会，各专家讨论与规模相关的因素。

（3）各专家匿名填写迭代表格。

（4）协调人整理出一个估计总结，以迭代表的形式将估计总结返回给专家。

（5）协调人召开小组会议，讨论较大的估计差异。

（6）专家复查估计，总结并在迭代基础上提交另一个匿名估计。

（7）重复步骤（4）～步骤（6），直到达到最低估计和最高估计的一致。

2）类比法

类比法适合评估一些与历史项目在应用领域、环境和复杂度方面相似的项目，通过新项目与历史项目的比较得到规模估计。类比法估计结果的精确度取决于历史项目数据的完整性和准确度。因此，用好类比法的前提条件之一，就是软件组织有强大的软件测量数据库和文档库，建立起了较好的项目评价与分析机制，使历史项目的数据分析是可信赖的。使用类比法的基本步骤如下。

（1）整理出项目功能列表和实现每个功能的代码行。

（2）标识出每个功能列表与历史项目的相同点和不同点，特别要注意历史项目做得不够的地方。

（3）通过步骤（1）和步骤（2）得出各个功能的估计值。

（4）产生规模估计。

软件项目中用类比法，还要进行可重用代码的估算。程序员或系统分析员详细地考查已存在的代码，估算出新项目可重用的代码中需要重新设计的百分比、需要重新编码或修改的百分比和需要重新测试的百分比。根据这三个百分比，用下面的计算公式计算等价新代码行：

等价代码行＝[（重新设计的百分比＋重新编码的百分比＋重新测试的百分比）/3]×已有代码行

例如，有 10 000 行代码，假定 30％需要重新设计，50％需要重新编码，70％需要重新测试，那么其等价的代码行为：

$$[(30\%+50\%+70\%)/3]\times 10\,000=5000(\text{等价代码行})$$

即重用这 10 000 代码相当于编写 5000 代码行的工作量。

3）功能点估计法

功能点（实体数、构件数、屏幕数、报表数、文档数）测量是在需求分析阶段基于系统功能的一种规模估计方法。通过研究初始应用需求来确定各种输入、输出、计算和数据库需求的数量和特性。执行功能点法的基本步骤如下所示。

(1) 计算输入、输出、查询、主控文件和接口需求的数目。

(2) 将这些数据进行加权相乘。一般输入、输出、查询、主控文件和接口需求的权值为4、5、4、10、10。

(3) 估计者根据对复杂度的判断,总数可以用+25%、0或-25%调整。

据统计发现,对一个软件产品的开发,功能点对项目早期的规模估计很有帮助。然而,在深入了解产品后,功能点可以被转换为软件规模估计更常用的LOC。

4) 无礼估计法

无礼估计法对各个项目活动的完成时间,按三种不同情况进行估计:一个产品的期望规模、一个产品的最低可能估计和一个产品的最高可能估计。用这三个估计得到一个产品期望规模和标准偏差的无礼统计估计。无礼估计法也可得到代码行的期望值E和标准偏差S_D。

2. 软件费用与资源估计方法

除了对软件规模和工作量进行估计之外,还要对软件成本费用与软件资源进行估计。只有这三个估计都完成后,软件估计才算最后完成。

1) 对软件工作产品成本费用进行量化估计

对软件工作产品成本费用进行量化估计,其方法如表5-4所示。

表5-4 软件工作产品成本费用估计的方法

序号	估计方法	估计单位	方法说明
1	直接劳务费	人民币元	开发人员的工资和福利
2	管理费	人民币元	技术管理人员和行政管理人员的工资和福利
3	差旅费	人民币元	售前、售中、售后的人员差旅费
4	计算机使用费	人民币元	网络设备的折旧费和房租水电费
5	其他招待费和公关费	人民币元	控制在总费用的15%以内

2) 对关键计算机资源进行量化估计

对关键计算机资源进行量化估计,其方法如表5-5所示。

表5-5 关键计算机资源估计的方法

序号	估计方法	方法说明
1	软件工作产品的规模	对存储能力(磁盘容量和内存大小)的要求
2	运行处理的负载	对处理器速度的要求
3	通信量	对网络通道和带宽的要求

5.4 软件策划管理与软件策划管理文档

1. 软件策划管理方法

软件策划管理的目的是建立对实际进展的可视性监控,使管理者能在计划发生明显偏离时采取有效措施。项目经理对策划进行管理的方法如下所示:

(1) 首先,按计划跟踪项目进度、软件工作产品规模和工作量、软件成本、关键计算机资源、软件工程技术活动和软件风险,并以此编制项目进展报告。

(2) 其次,定期地或以事件驱动地组织软件工程组进行内部评审,并将评审结果告知软件相关组。当软件开发计划发生20%以上的偏离时,必须提出软件开发计划变更申请,经评审和批准后,修改《软件开发计划》,并将修改的结果通知有关的组和个人。

2. 软件策划管理文档

软件策划管理的输入是《软件开发计划》和项目组的实际工作进度与状态。

软件策划管理的输出文档是:

- 项目周报
- 项目月报
- 里程碑报告
- 重大事件报告
- 软件开发计划书评审报告
- 项目计划变更申请书
- 计划更改与批准记录

3. 软件开发计划书

软件开发计划也被称为软件项目计划(Software Project Planning),是指在正式进行软件开发之前,制定的具体指导软件开发的实施计划,是指导软件开发工作的纲领。制定软件开发计划的目的是以文件的形式,对开发过程中各项工作的负责人员、开发进度、所需经费预算、所需软硬件条件等事项进行安排,以便实施和检查项目的开发工作。

1) 软件开发计划的主要内容

《软件开发计划书》就是软件策划产生的文档,具体内容可参考国家标准GB-T 8567-2006《计算机软件文档编制规范》,可结合项目的规模、类型、条件等特点进行适当调整。

主要包括以下内容。

(1) 项目概述。简单说明项目各项主要工作内容;概述软件的功能、性能和可靠性;完成项目应具备的条件;用户及合同承包者承担的工作及技术水平;完成期限及验收标准;应交付的程序名称,所使用的语言及存储形式;应依附的文档;向用户提供的各项服务,包括培训安装、维护、运行支持、服务期限等。

(2) 实施计划。包括:各阶段具体任务的划分,各项任务的责任人;项目开发进度,按阶段应完成的任务,用图表说明每项任务的开始时间和完成时间;项目的预算,各阶段的费用支出预算;关键问题、技术难题和风险及影响。

(3) 人员组织及分工。包括:开发该项目所需人员的类型、组成结构、数量等。

(4) 交付期限。主要指软件项目最后完工交付的具体日期。

2) 软件开发计划书

软件开发计划书的编写格式和要求可参考以下内容。

1. 引言(Introduction)

1.1　目的(Purpose)

本章提供对整个软件开发计划的综述。主要是确定以下内容。

(1) 软件生命周期的选择与裁剪。

(2) 确定软件开发和维护的规范、方法和标准。

(3) 软件工作产品的规模和工作量估计。

(4) 软件成本和计算机资源的估计。

(5) 软件进度表的制定。

(6) 软件风险的估计。

(7) 软件项目培训计划。

1.2　范围(Scope)

说明该软件开发计划的范围,简要描述软件开发计划的内容。一般而言,对于一个较大的软件项目(工期6个人月以上),计划书将包括如下内容。

(1) 软件规模估计。

(2) 工作模块计划。

(3) 人力资源计划。

(4) 其他资源计划。

(5) 进度安排计划。

(6) 配置管理计划(可单独做一个计划)。

(7) 质量保证计划(可单独做一个计划)。

1.3　术语定义(Terms Glossary)

对该软件开发计划中的术语、缩写词进行定义。包括用户应用领域与计算机领域的术语与缩写词等。例如:

(1) 软件相关组:指软件配置管理组、文档支持组、测试组。

(2) 软件质量保证组:指计划和实施软件质量保证活动的人员的集合。

1.4　参考资料(References)

说明该软件开发计划使用的参考资料,如项目的用户需求报告、商务合同、用户领域的资料等,每一个文件、文献要有标题、索引号或文件号,发布或发表日期以及出版单位。

1.5　相关文档(Related Documents)

当该文档变更时,可能对其他文档产生影响,受影响的文档叫做相关文档,需要将它们列出。

1.6　版本更新记录(Version Updated Record)

版本更新记录格式,如表5-6所示。

表5-6　版本更新记录

版本号	创建者	创建日期	维护者	维护日期	维护纪要
V1.0	李东生	2012/02/06	—	—	—
V1.0.1	—	—	王海林	2012/02/18	成本估算维护
…					

2. 项目概述(Project Summary)

2.1 项目的目的(Project Purpose)

说明该软件项目的目的。

2.2 项目的范围(Project Scope)

本章的内容,主要参照《立项建议书》或《合同》与《用户需求报告》中的相关章节,简要描述该软件项目的实现范围。

(1) 主要功能点列表。

(2) 主要性能点列表。

(3) 主要接口列表。

(4) 本软件项目与其他软件项目之间的关系。

(5) 项目实施方面的限制等内容。

2.3 项目的使用对象(Project Reader)

在本章节中,要识别出顾客与最终用户,对顾客与最终用户的情况要有简单描述,如最终用户的教育水平、技术水平、本系统的使用频率等。

3. 项目组织(Project Organization)

项目组织是为开发项目而组建的队伍。建议以框图的方式表示项目的组织结构,并对每一组织的负责人和职责加以说明。可能的项目组织单元包括项目管理组、质量保证组、配置管理组、软件工程组、测试组和需求管理组。各组织的说明如下所示。

(1) 项目管理组:对项目实施负全部责任。

(2) 质量保证组:负责项目过程与产品的质量控制和报告。

(3) 配置管理组:负责项目产品的版本、配置管理以及配置库状态报告。

(4) 软件工程组:执行软件项目工程过程,负责项目产品的开发和维护工作。

(5) 测试组:执行软件项目测试过程,负责项目产品的测试。

(6) 需求管理组:负责对需求基线和需求变更进行管理。

4. 软件生命周期(Software Life Cycle)

本章节记录项目策划生命周期定义的工作结果,需要描述的主要内容如下所示。

(1) 项目生命周期框图。

(2) 项目生命周期说明。

5. 规范、方法和标准(Criterion、Means and Standard)

本节需要描述采用的供开发和维护用的规范、方法和标准。

6. 任务与工作产品(Task and Work Products)

项目任务和工作产品,是指根据项目生命周期阶段划分的任务,和相应阶段的工作产品。记录项目生命周期各阶段确定的需重点控制的阶段任务和工作产品。建议以表格的形式,列出生命周期各阶段的任务和工作产品。项目包含的任务包括需求分析、系统设计、系统实现、测试、产品交付和产品维护。

项目可能包含的产品包括需求分析说明书、规格分析说明书、系统设计说明书、源代码、各种测试报告、用户手册和软件问题维护记录。

7. 工作产品、任务规模和工作量估计（Estimates of Work Product, Task Size and Workload）

项目规模估算是为了确定项目所需的人工。需要描述的主要内容有以下三个。

(1) 对软件工作产品规模估计依据的简要描述。

(2) 对每种任务和工作产品规模估计的结果。

(3) 规模估算的结果，建议用《任务规模和工作量估算表》的形式列出。

8. 成本估计（Estimates of Coats）

成本估计是指对项目完成过程中耗费的人力、物力、财力资源的估算。成本估计应按类别进行估算，可能的成本估算类别包括直接人工、直接费用、间接成本、制造费用、管理费用和不可预见费用。

9. 关键计算机资源计划（Critical Computer Resource Plan）

项目的关键计算机资源计划是指系统在开发环境、测试环境及用户目标环境中，对关键计算机资源，如计算机存储能力、计算机处理器速度、通信通道容量、服务器处理能力等的估计，使之能满足软件开发、测试和运行的要求。

10. 软件项目进度（Software Project Schedule）

软件项目进度计划，是对项目的进度、人员工作分工所做的计划，此计划依据上述各章的估算和分析结果，计划方式建议采用表格的形式。若采用工具制定项目计划，应将工具生成的图表作为项目计划的附件。本章节中需要描述的主要内容有以下四个。

(1) 软件项目每个阶段的进度时间表。

(2) 设定的里程碑。

(3) 评审时间。

(4) 缓冲时间。

11. 配置管理计划（Configuration Management Planning）

可单独做一个计划。

12. 质量保证计划（Software Quality Assurance Planning）

可单独做一个计划。

13. 风险分析（Risks Analysis）

风险分析是对可能发生的将会对项目按预期时间、资源和预算完成产生重大影响的事件的分析，包括以下内容。

(1) 被识别出的重大风险事件：政策风险、技术风险、技能风险等。

(2) 易发生重大风险事件的高风险区域：用户需求、设计、测试、运行平台等。

(3) 重大风险事件的级别：功能不全、性能不稳、速度受限制等。

(4) 拟采取的预防措施：增加投入、纠错、延时等。

(5) 风险事件发生后建议采用的处理措施：更改计划、降低系统难度等。

14. 设备工具计划（Equipment and Tools Planning）

项目设备工具计划是根据项目的工作指派及进度确定项目所需要的设备和工具，以确保设备工具在任务执行前到位，保证项目任务的顺利执行，在本计划中应包含以下几方面的

内容：所需的设备、基本的要求和应到位的时间。

15. 培训计划(Training Planning)

应根据项目的特点和项目组成员技能情况，制定出项目组成员所需的培训内容，培训计划中应包含以下几方面：培训内容、培训时间、教员、接受培训的人员和培训目的(应达到的效果)。

16. 项目评审(Project Reviews)

项目评审是对项目策划过程所做的定期性评审。其内容可分为：评审点、评审周期、评审层次、评审条款和措施、管理评审活动中提交的工作产品(列出被评审的工作产品)。

17. 度量(Measurement)

度量是按规定在项目进行过程中，需要采集的度量数据，以便量化地反映项目的进展情况，为管理者提供对项目进展的适当的可视性，同时度量数据是项目过程改善的数据基础。应规定项目度量值的记录人(一般为项目经理或其指定人员)、记录时间(一般以定期评审为基础)和记录的数据。常用的度量数据如下所示。

(1) 项目过程的评审次数。

(2) 项目计划修改次数。

(3) 项目各阶段的人员投入(各阶段投入人的人月数)。

(4) 各类任务耗用时间统计(如设计、编码、测试、文档编写等)。

(5) 工作产品统计(如文档字数、功能点数、用例数、源代码行数等)。

5.5 本章小结

软件策划既是为软件开发者和管理者制定合理的计划，又是为软件项目跟踪和监控提供考核依据。软件策划是项目经理和高级经理的职责范围，是IT企业的重大事件之一。软件估计既是软件策划的核心，又是软件策划的重点与难点。如果说软件立项就是软件组织的重大决策，那么软件策划就是贯彻执行重大决策的具体行动。立项或签订合同是软件项目的源头，策划是指导软件项目开发和管理的依据。

通过对本章的学习，应该清楚地知道：软件策划的目的，策划要实现的具体目标，软件策划的时机，策划的输入文档和输出文档，基本策划方法，软件估计的内容和方法，策划的具体过程，软件开发计划书的内容，软件开发计划的制作格式，软件策划管理的方法。

为了使软件策划有坚实的基础，使软件开发计划不至于过大地偏离项目工程进度、质量、资源的实际(小于20%)，最常用的办法是：

- 策划的时机选择在《用户需求报告》制定之后和《需求规格说明书》制定之前。
- 软件估计时查阅软件组织的软件过程数据库(或软件测量数据库)，应参照同类可比项目的历史经验。
- 由同行专家对《软件开发计划书》进行评审。

习题 5

1. 简述软件策划的过程。
2. 软件策划要实现的具体目标是什么?
3. 为什么在策划过程中要考虑受影响的组和个人?
4. 怎样理解对软件项目进行策划的时机?
5. 简述软件策划的方法。
6. 软件估计的含义是什么?
7. 简述对软件工作产品规模进行量化估计的方法。
8. 简述对软件工作产品成本费用的估计方法。
9. 项目跟踪与监督的基础是什么?
10. 软件开发计划应包括哪些内容?
11. 请设计出以下策划管理文档:项目周报、项目月报、里程碑报告、重大事件报告、软件开发计划评审报告、项目计划变更申请表和计划更改与批准记录。
12. 在老师的指导下编写一份图书馆信息系统的《软件开发计划书》。

第6章 软件管理

6.1 三个模型的建模思想

软件开发的主要工作是软件需求和软件设计，软件需求和软件设计的关键问题是软件建模，简称建模。

所谓模型，就是为了理解事物而对事物所做的一种抽象，是对事物的一种无歧义的书面描述。模型通常是由一组图示符号和组织这些符号的规则组成，可以利用它们来定义和描述问题域中的术语和概念。更进一步讲，模型是一种思考工具，利用这种工具可以把知识规范地表示出来。也就是说，模型能让系统构造者用标准的、易于理解的方式表达他们对系统的设想，并且提供了一种便于人与人之间有效地共享和交流设计结果的机制。通过建模，可达到以下四个目的。

- 模型帮助设计人员按照实际情况或所需要的样式对系统进行可视化。
- 模型允许详细说明系统的结构或行为。
- 模型给出一个指导构造系统的模板。
- 模型对做出的决策进行文档化。

为了开发复杂的软件系统，系统分析员应该从不同角度抽象出目标系统的特性，使用精确的表示方法构造系统的模型，验证模型是否满足用户对目标系统的需求，并在设计过程中逐渐把和实现有关的细节加进模型中，直至最终用程序实现模型。对于那些因过分复杂而不能直接理解的系统，尤其需要建立模型，其目的是为了减少复杂性。人的头脑每次只能处理一定数量的信息，模型通过把系统的重要部分分解成人的头脑一次能处理的若干个子部分，从而减少了系统的复杂程度。在对目标系统进行分析的初始阶段，面对大量模糊的、涉及众多专业领域的、错综复杂的信息，系统分析员往往会感到无从下手，而模型则对此提供了组织大量信息的一种有效机制。

一旦建立起模型之后，这个模型就要通过用户和各个领域专家的严格审查。由于模型的规范化和系统化，因此可以比较容易地暴露出系统分析员对目标系统认识的片面性和不一致性。通过审查，往往会发现许多错误，这是正常现象，这些错误可以在成为目标系统中的错误之前，被预先修正。

通常，用户和领域专家可以通过快速建立的原型亲身检验，从而对系统模型进行更有效的审查。模型常常会经过多次必要的修改，通过不断改正错误的认识或不全面的认识，最终使软件开发人员对问题有了透彻的理解，从而为后续的开发工作奠定了坚实的基础。

用面向对象方法成功地开发软件的关键，同样是对问题域的理解。面向对象方法最基本的原则是按照人们习惯的思维方式，用面向对象的观点建立问题域的模型，开发出尽可能自然地表现求解方法的软件。用面向对象方法开发软件，通常需要建立三种形式的模型，它们分别是描述系统数据结构的对象模型、描述系统控制结构的动态模型和描述系统功能的功能模型。这三种模型都涉及数据、控制、操作等共同的概念，只不过每种模型描述的侧重点不同。三种模型从三个不同但又密切相关的角度模拟目标系统，各自从不同侧面反映了系统的实质性内容，综合起来则全面地反映了对目标系统的需求。

6.1.1 对象模型

对象模型表示静态的、结构化的、系统的“数据”性质，是对模拟客观世界实体的对象以及对象彼此之间的关系的映射，描述了系统的静态结构。在三个模型中，对象模型始终都是最重要、最基本、最核心的，它为建立动态模型和功能模型提供了实质性的框架，人们依靠对象模型完成三个模型的集成。对象模型是一个类、对象、类和对象之间关系的定义集，它把面向对象的概念与传统方法中常用的信息建模概念结合起来，改进和扩展了普通的信息模型，增强了模型的可理解性和表达能力。在需求分析中，对象模型既可以用来表示系统数据，也可以用来表达对数据的处理，可以看做是数据流和语义数据模型的结合。

对象模型本身是静态的，但是在设计者心目中，应该尽量将它由静态变成动态。设计者可以想象数据(或记录)在相关表上的流动过程，即增加、删除、修改、传输、处理等，从而在脑海中运行系统，或在 E-R 图上运行系统。

6.1.2 动态模型

动态模型表示瞬时的、行为化的系统“控制”性质，它规定了对象模型中对象的合法变化序列。

一旦建立起对象模型之后，就需要考察对象的动态行为。所有对象都具有自己的生命周期(或称为运行周期)。对一个对象来说，生命周期由许多阶段组成，在每个特定阶段中，都有适合该对象的一组运行规律和行为规则，用以规范该对象的行为，生命周期中的阶段也就是对象的状态。所谓状态，是对对象属性值的一种抽象，在定义状态时应该忽略那些不影响对象行为的属性。各对象之间相互触发(即作用)就形成了一系列的状态变化。人们把一个触发行为称做一个事件。对象对事件的响应，取决于接受该触发的对象当时所处的状态，响应包括改变自己的状态或者又形成一个新的触发行为。

状态有持续性，它占用一段时间间隔。状态与事件密不可分，一个事件分开两个状态，一个状态隔开两个事件，事件表示时刻，状态代表时间间隔。

每个类的动态行为用一张状态图来描绘，各个类的状态图通过共享事件合并起来，从而构成系统的动态模型。也就是说，动态模型是基于事件共享而互相关联的一组状态图的集合。

6.1.3 功能模型

功能模型表示变化的系统的“功能”性质，指明了系统应该“做什么”，更直接地反映了用

户对目标系统的需求。因此，功能模型实质上是用户需求模型，反映了系统的功能需求，对系统的功能、性能、接口和界面进行定义，是用户界面模型设计的主要依据。功能模型既是动态的又是静态的，因为有的功能与系统运行时间序列有关。功能模型既是数据库和数据结构设计的基础，又是功能模块设计、编程实现和测试验收的依据。

通常，功能模型由一组数据流图组成。在面向对象方法学中，数据流图远不如在结构分析、设计方法中那样重要。一般说来，与对象模型和动态模型比较起来，数据流图并没有增加新的信息，但是，建立功能模型有助于软件开发人员更深入地理解问题域，改进和完善自己的设计。因此，不能完全忽视功能模型的作用。

功能模型的设计和实现方法为：将相同的功能归并，设计为一个个的构件或组件（部件），将不同的功能设计成模块，然后用面向对象的语言将这些离散的部件或模块组装起来，形成一个完整的系统。

6.1.4 三个模型之间的关系

面向对象建模技术所建立的三个模型，分别从不同侧面描述了所要开发的系统。这三个模型相互补充、相互配合，使得人们对系统的认识更加全面：功能模型指明了系统应该“做什么”；动态模型明确规定了什么时候（即在何种状态下接受了什么事件的触发）做；对象模型则定义了做事情的实体。三个模型之间的关系描述如下：

- 针对每个类建立的动态模型，描述了类实例的生命周期或运行周期。
- 状态转换驱使行为发生，这些行为在数据流图中被映射成处理，在用例图中被映射成用例，它们同时与类图中的服务相对应。
- 功能模型中的处理（或用例）对应于对象模型中的类所提供的服务。通常，复杂的处理（或用例）对应于复杂对象提供的服务，简单的处理（或用例）对应于更基本的对象提供的服务。有时一个处理（或用例）对应于多个服务，也有一个服务对应于多个处理（或用例）的时候。
- 数据流图中的数据存储，以及数据的源点或终点，通常是对象模型中的对象。
- 数据流图中的数据流，往往是对象模型中对象的属性值，也可能是整个对象。
- 用例图中的行为者，可能是对象模型中的对象。
- 功能模型中的处理（或用例）可能产生动态模型中的事件。
- 对象模型描述了数据流图中的数据流、数据存储以及数据源点或终点的结构。

6.2 数据模型设计概论

6.2.1 数据模型

数据模型是对现实世界中各种事物或实体特征的数字化模拟和抽象，用以表示现实世界中的实体及实体之间的联系使之能存放到计算机中，并通过计算机软件进行处理的概念工具的集合。

数据模型三要素如下所示。

1）数据结构

用于描述现实系统中数据的静态特性，是对象类型的集合，包括与数据类型、内容、性质有关的对象；与数据之间联系有关的对象。

2）数据操作

用于描述数据的动态特性，是对数据库中各种对象（型）的实例（值）允许执行的操作及有关的操作规则，其类型包括检索和更新（包括插入、删除和修改）。

数据模型对操作的定义包括：

- 操作的确切含义
- 操作符号
- 操作规则（如优先级）
- 实现操作的语言

3）数据约束

数据的约束条件用于描述对数据的约束，它是一组完整性规则的集合。完整性规则是给定的数据模型中数据及其联系所具有的制约和储存规则，用以限定符合数据模型的数据库状态以及状态的变化，以保证数据的正确、有效和相容。

数据模型对约束条件的定义为：

- 反映和规定本数据模型必须遵守的、基本的、通用的完整性约束条件。例如在关系模型中，任何关系必须满足实体完整性和参照完整性两个条件。
- 提供定义完整性约束条件的机制，以反映具体应用所涉及的数据必须遵守的、特定的语义约束条件。

数据模型可分为以下三类。

1. 概念数据模型

概念层次的数据模型称为概念数据模型，简称概念模型。概念模型离机器最远，从机器的立场上看是抽象级别的最高层，目的是按用户的观点或认识来对现实世界建模，因此它应该具备以下特点：

- 语义表达能力强
- 易于用户理解
- 独立于任何 DBMS
- 容易向 DBMS 所支持的逻辑数据模型转换

2. 逻辑数据模型

逻辑层是数据抽象的中间层，用于描述数据库数据的整体逻辑结构。这一层的数据抽象称为逻辑数据模型（简称数据模型）。它是用户通过数据库管理系统看到的现实世界，是数据的系统表示，即数据的计算机实现形式。因此它既要考虑用户容易理解，又要考虑便于 DBMS 实现。不同的 DBMS 提供不同的逻辑数据模型，传统的数据模型有层次、网状、关系模型，非传统的数据模型有面向对象数据模型（简称 OO 模型）。

3. 物理数据模型

物理层是数据抽象的最低层,用来描述数据物理存储结构和存储方法。例如一个数据库中的数据和索引是存放在不同的数据段上还是相同的数据段上;数据的物理记录格式是变长的还是定长的;数据是压缩的还是非压缩的;索引结构是 B+树还是 HASH 结构等。这一层的数据抽象称为物理数据模型,它不但由 DBMS 的设计决定,而且与操作系统、计算机硬件密切相关。

数据模型是数据抽象的工具,是数据组织和标识的方式;数据模式是数据抽象利用数据模型,将数据组织起来后得到的结果,如图 6-1 所示。

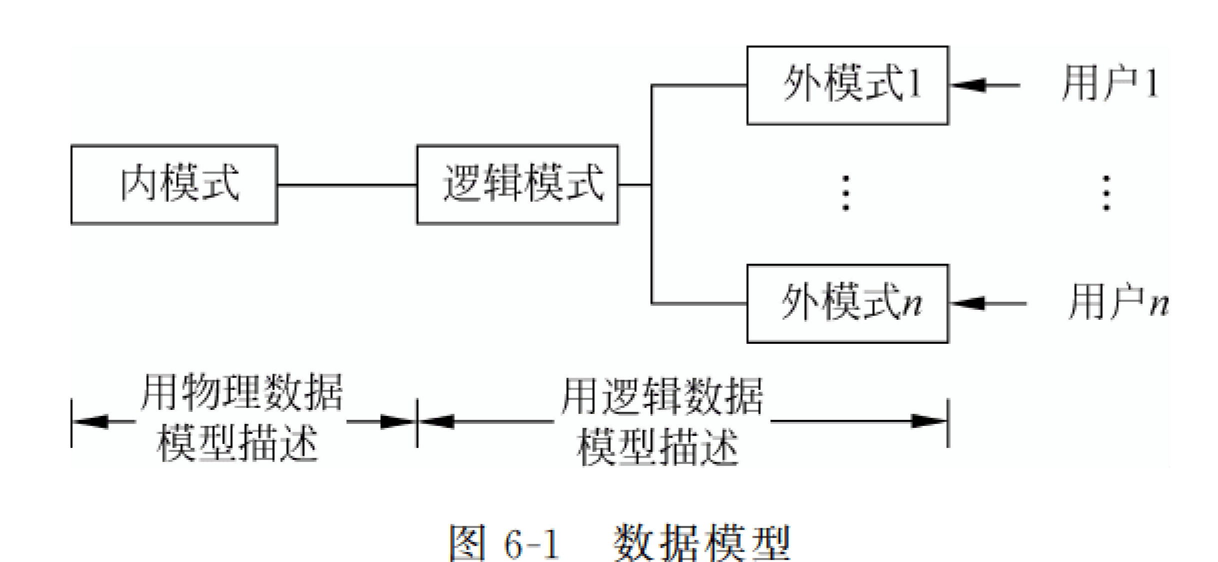


图 6-1 数据模型

6.2.2 概念数据模型

为了把现实世界的事物抽象为数据库管理系统支持的数据模型,人们必须首先把现实世界抽象为信息世界,而后再将信息世界映像为机器世界。存在于人们头脑之外的客观世界,被称为现实世界。如在图书馆藏书和管理系统中,有图书、借书证、学生和教师、图书管理员、借阅规则、罚款规则等。此外,还有各种查询报表、统计报表等。把这些客观存在的事实收集起来,进行分类,抽取系统所需要的数据。人们用文字、图形、符号等表示现实世界在头脑中的反映,构成信息世界。在信息世界中,数据库技术通常用到下列术语:实体、实体集、属性、联系、实体标识等。由于计算机只能处理数据化的信息,所以对信息世界中的信息必须进行数据化,数据化后的信息称为数据。所以信息世界的信息在计算机系统中以数据形式存储。机器世界中数据描述的术语主要有:字段、记录、文件、键等。

以教学管理系统为例,系统中有教师、课程、学生等,在表 6-1 中简单列出了它们的差别。

表 6-1 概念数据模型

现 实 世 界	信 息 世 界	机 器 世 界
教师、课程和学生	教师实体、课程实体和学生实体	数据记录(Record)
教师集合、学生集合和课程集合	教师实体集、学生实体集和课程实体集(Entity set)	数据记录集(Data Set)
教师、课程和学生的特征,如编号、名称等	教师实体、课程实体和学生实体的属性(Attribute)	数据项(Field 或 Item)
用以区分对象的特征	实体标识	码
对象之间的关系	联系	地址或数据项

1. 实体-联系(E-R)模型

概念模型最常见的描述方法是采用图形化方法，这种方法直观、自然，易于描述系统的层次结构、功能组成，且简单易学，通常还有工具软件支持，因而成为信息系统的主要描述工具。最著名的图示化描述方法是“实体-联系方法”，简称 E-R 方法。用这种方法建立的概念模型称为 E-R 模型(见图 6-2)。Erwin 是用来建立 E-R 模型的 CASE 工具，主要用于 E-R 模型的设计和数据库模式的设计。

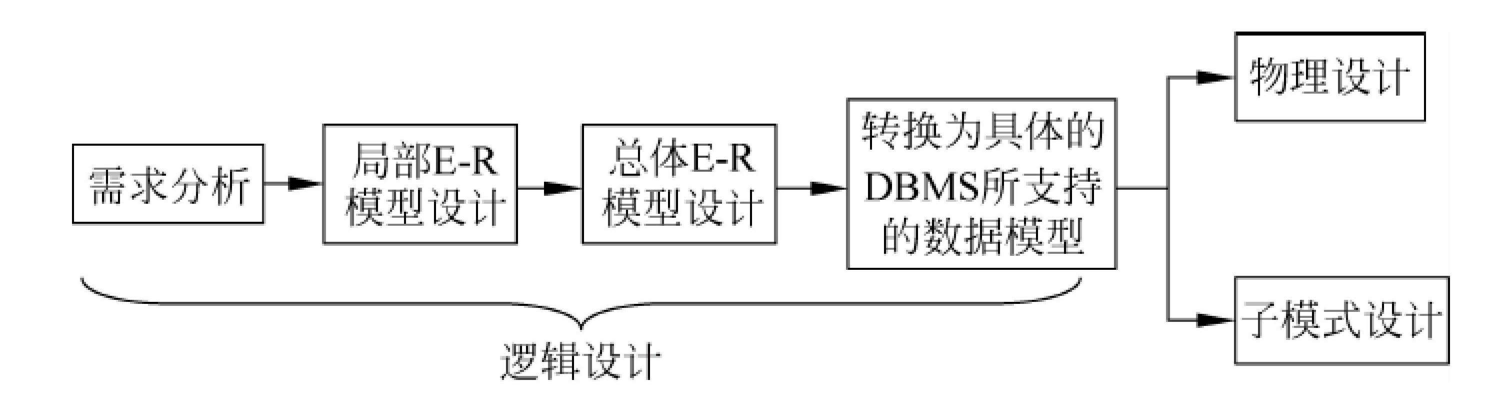


图 6-2 实体-联系(E-R)模型

1) E-R 模型的三个组成部分

- 实体集：具有相同类型和性质(属性)的实体集合。
- 联系集：同类联系的集合。
- 属性：实体的性质和特征。

2) E-R 模型的表示

- 实体用方框表示，联系用菱形框表示，属性用椭圆框表示(见图 6-3)。
- 在框中标注实体名、联系名和属性名。

用以上三种图形元素描述的概念模型称为 E-R 图，它可以看成是描述数据库概念模型的图形语言，实体是名词、属性是名词、联系是动词。

实体名 联系名 属性名

图 6-3 实体联系属性

3) 画 E-R 图的方法(见图 6-4)

- 先画出各实体及其属性，其中的主属性(即主键)用下划线表示。
- 在有联系的实体之间插入联系框，并用无向边把它们连接起来(联系也可能包含属性)。
- 在无向边上标注联系的类型。

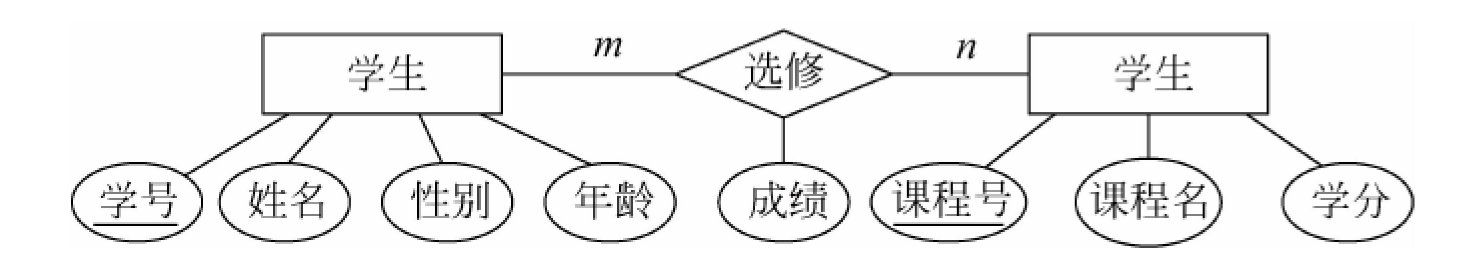


图 6-4 画 E-R 图的方法

4) E-R 图中的基本联系方式(见图 6-5)

二元联系(两实体间的联系)：包括一对一、一对多、多对多三种类型。

两实体间的多个联系(见图 6-6)：一个职工可参加多项工程，一项工程由多个职工参加；一个职工可负责多项工程，一项工程只有一个负责人。

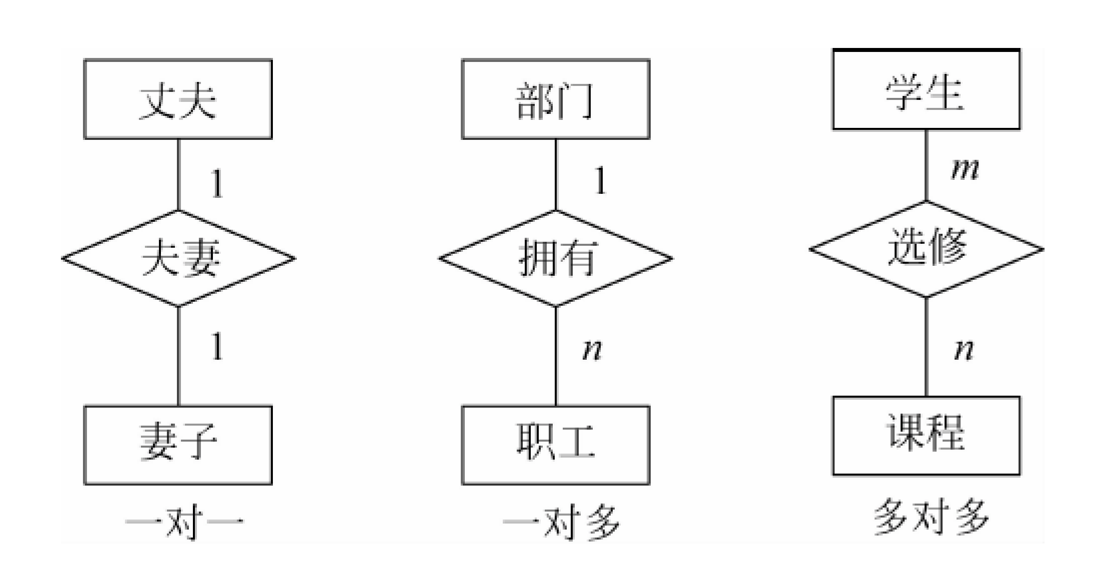


图 6-5　E-R 图的基本联系方式

实体内部的联系：一个零部件可由多个其他零部件组成；一个零部件可以是多个其他零部件的组成部分。

多元联系(两个以上实体间的联系)(见图 6-7)：如一个制片公司可以为一部电影与多名演员签约；一名演员可以与一个制片公司签约演出多部电影；一部电影可以由一个制片公司与多名演员签约演出。

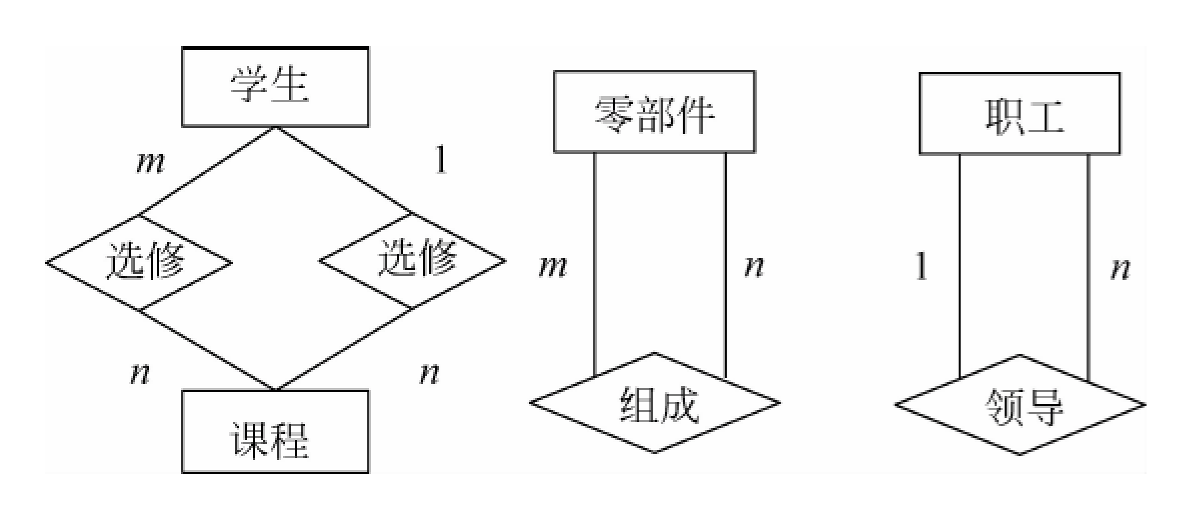


图 6-6　E-R 两实体间的多个联系

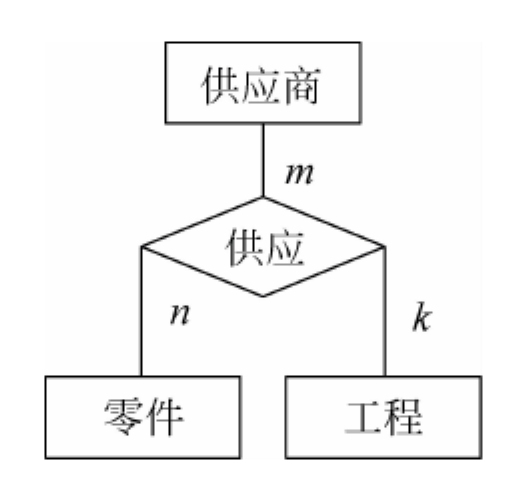


图 6-7　多元联系

多元联系可分解为多个二元联系，如图 6-8 所示。

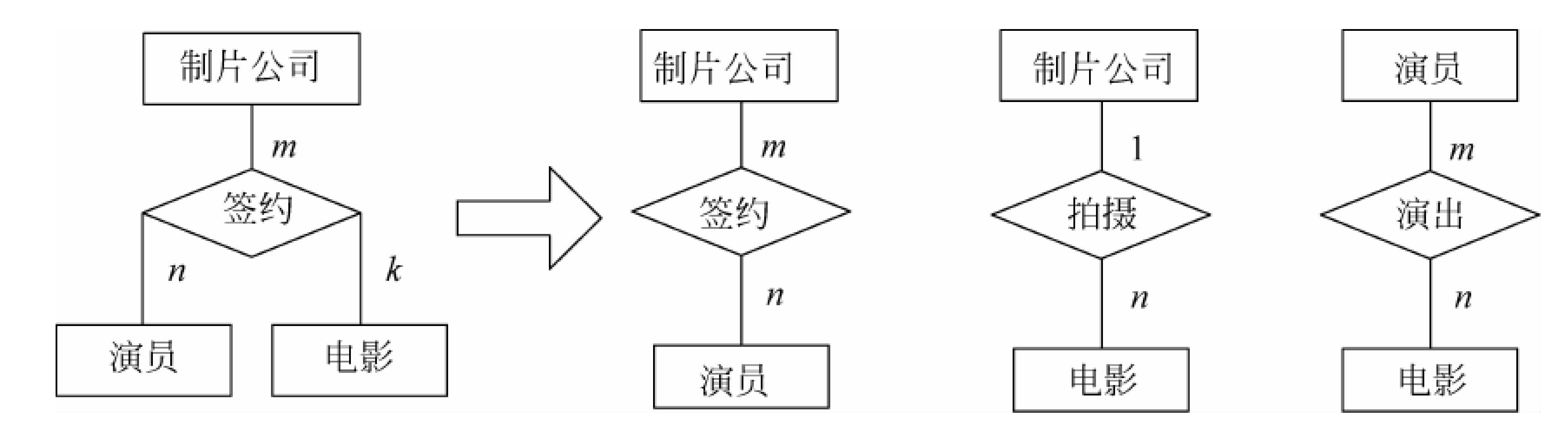


图 6-8　多元联系分解

5）联系的属性

联系也是一种关系，它也可能拥有属性，例如假设上例中的“签约”需要记录各演员的演出酬金。

- 酬金不是电影的属性：不同的演员有不同的酬金。
- 酬金不是制片公司的属性：不同的演员或电影有不同的酬金。
- 酬金不是演员的属性：不同的电影或制片厂有不同的酬金。

所以酬金应作为“签约”联系的属性。

6）子类和父类

- 子类除共享父类的公共属性外，它还有自己的特殊属性。
- 父类是子类的泛化实体，它具有其下属的所有子类的公共属性。

2．如何设计 E-R 模型

1）E-R 模型要在需求分析的基础上进行设计

- 根据需求分析确定实体、属性和实体间的联系
- 设计局部 E-R 模型
- 对局部 E-R 模型进行综合，设计出总体 E-R 模型
- 消除冗余，优化总体 E-R 模型

2）E-R 模型的设计三原则

- 相对原则：E-R 模型没有绝对的结果，不同的设计者基于不同的考虑所建立的模型很可能不同，但只要能满足业务的需求，它们都是正确的。
- 简单原则：现实世界的事物能作为属性对待的，尽量作为属性处理，而不要作为实体处理（设计局部 E-R 模型时应考虑的问题）。
- 一致原则：同一对象在不同的业务子系统（即局部 E-R 模型）中的抽象结果要求保持一致（对局部 E-R 模型进行综合时应考虑的问题）。

3）如何确定一个事物是属性还是实体

应根据应用环境来确定——是否关心该事物的“细微结构”。例如课程和教室：若不需要考虑教室的特性，则教室可作为课程的属性；否则就应把教室作为实体（见图 6-9）。

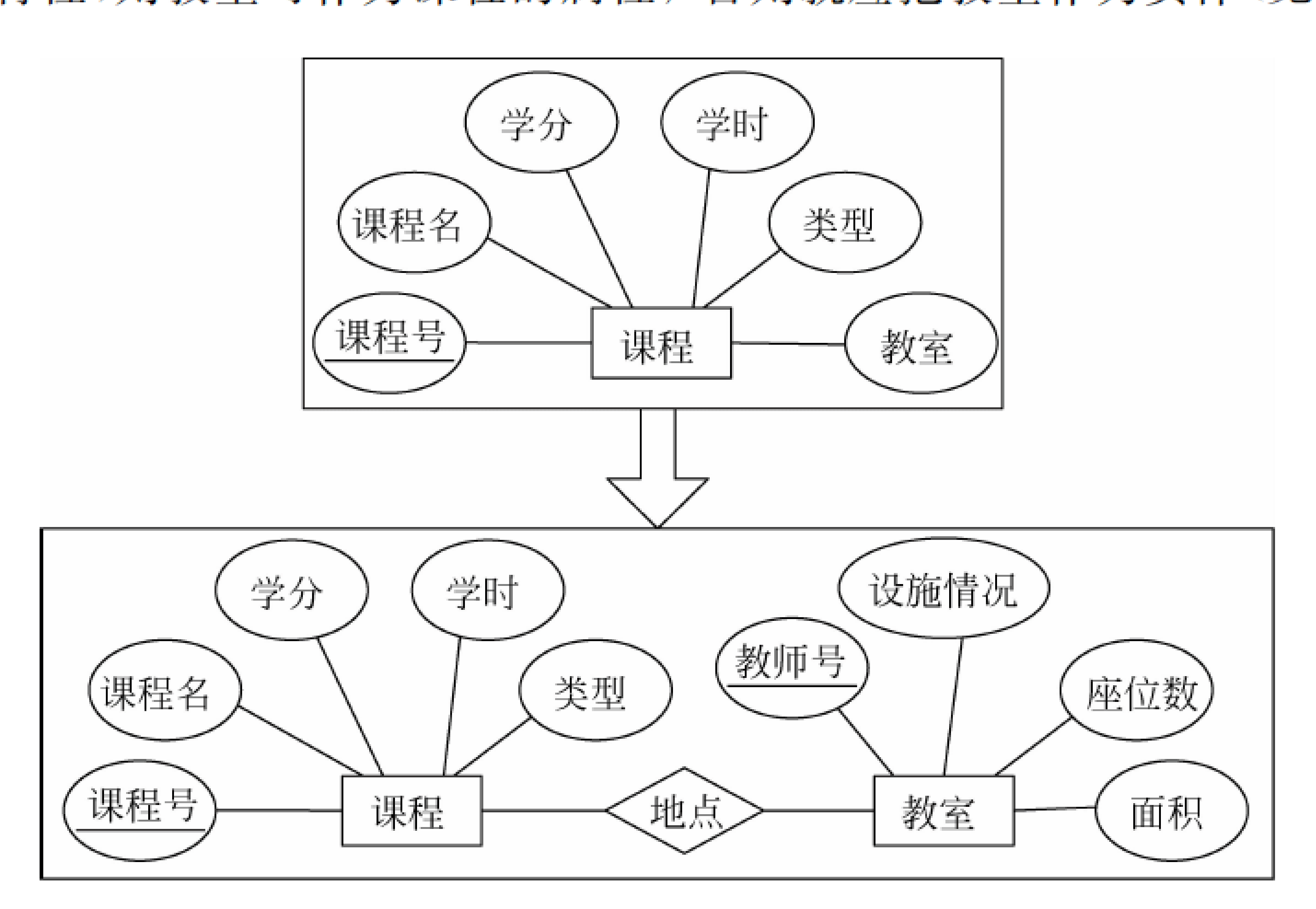


图 6-9　局部 E-R 图

4）E-R 模型的设计步骤

E-R 模型设计的要点如下所示。

- 化整为零：进行子系统划分，即把整个应用系统分为若干个相对独立的应用，这样就可以对每一个子系统分别进行设计，得出局部 E-R 模型。
- 化零为整：把局部 E-R 模型进行整合，设计出总体 E-R 模型。

E-R 模型的设计步骤分为三个阶段：

- 设计局部 E-R 模型。关键是确定子系统中有哪些实体，实体又包含哪些属性，它们之间有什么联系。
- 设计总体 E-R 模型。对局部 E-R 模型进行综合：对相同实体进行合并；为属于不同的局部 E-R 模型的实体间建立联系。
- 优化总体 E-R 模型。消除由于合并带来的冗余和冲突。

3. E-R 模型的特点

E-R 数据模型作为语义数据模型，是软件工程和数据库设计的有力工具，综合 E-R 数据模型的特点如下：

(1) 有丰富的语义表达能力，能充分反映现实世界，包括实体和实体间的联系，能满足用户对数据对象的处理要求。

(2) 易于交流和理解，因为它不依赖于计算机系统和具体的 DBMS，所以是 DBA、系统开发人员和用户之间的桥梁。

(3) 易于修改和扩充。

(4) 易于向其他各种数据模型(层次、网状、关系模型)转换。

(5) 实体、属性和联系这三个概念是有明确区分的，但对于某个具体的数据对象，究竟是作为实体，还是作为属性或联系，则是相对的，这取决于应用背景和用户的观点。

6.2.3　逻辑数据模型

1. 层次数据模型

层次数据模型是用树形结构来表示实体间联系的模型，如图 6-10 所示。

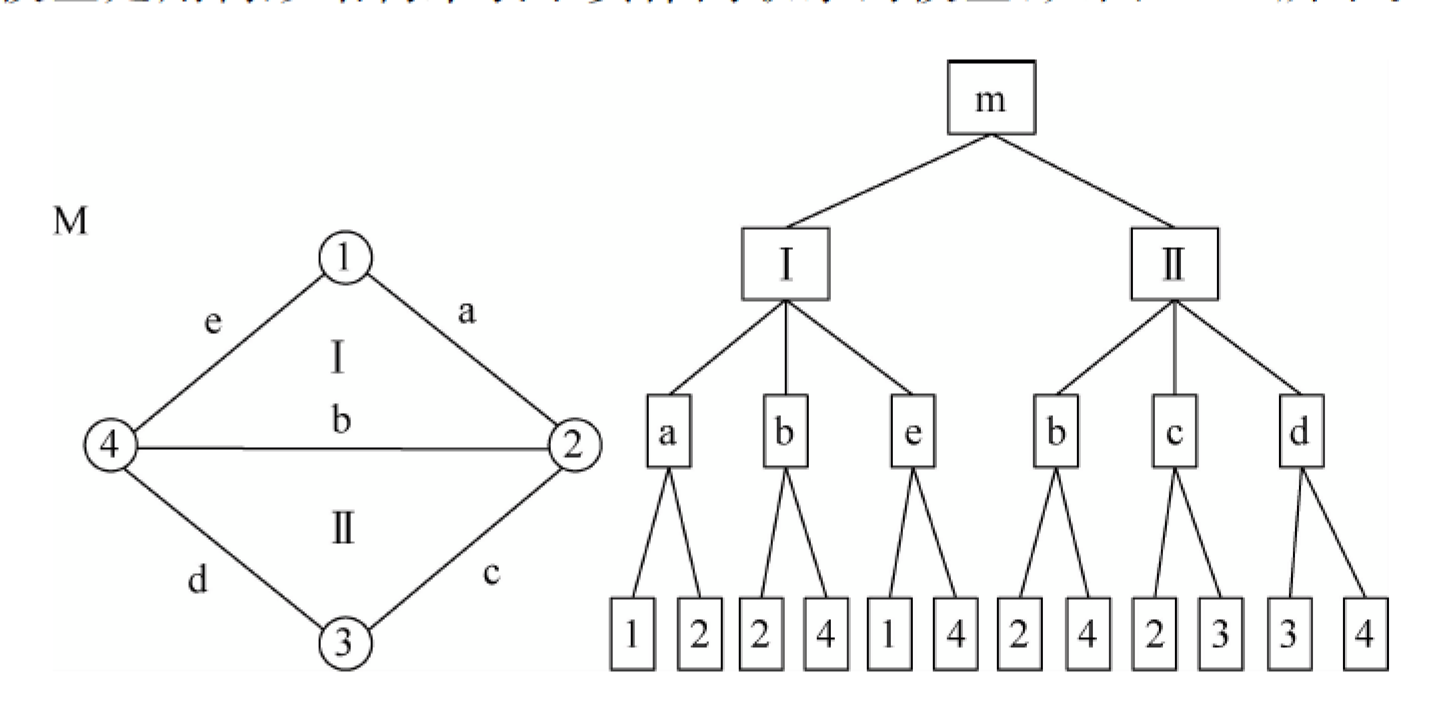


图 6-10　层次数据模型树形结构

1) 层次数据模型的特点

- 结点的双亲是唯一的。
- 只能直接处理一对多的实体联系。
- 每个记录类型定义一个排序字段，也称为码字段。
- 任何记录值只有按其路径查看时，才能显出它的全部意义。
- 没有一个子女记录值能够脱离双亲记录值而独立存在。

2）层次数据模型的优点

- 层次数据结构比较简单清晰。
- 层次数据库的查询效率高。
- 层次数据模型提供了良好的完整性支持。
- 结构严密，层次命令趋于程序化。

层次模型对具有一对多的层次联系的部门描述非常自然直观、容易理解，这是层次数据模型的突出优点。

3）层次数据模型的缺点

- 现实世界中的很多联系是非层次性的，如结点之间具有多对多联系。
- 对于一个结点具有多个双亲的联系，层次模型表示的方法很笨拙，只能通过引入冗余数据或创建非自然的数据结构来解决。
- 对插入和删除操作的限制比较多，因此应用程序的编写工作比较复杂。
- 查询子女结点必须通过双亲结点。

2．网状数据模型

用网状结构来表示实体及其联系的模型就是网状模型，如图 6-11 所示。

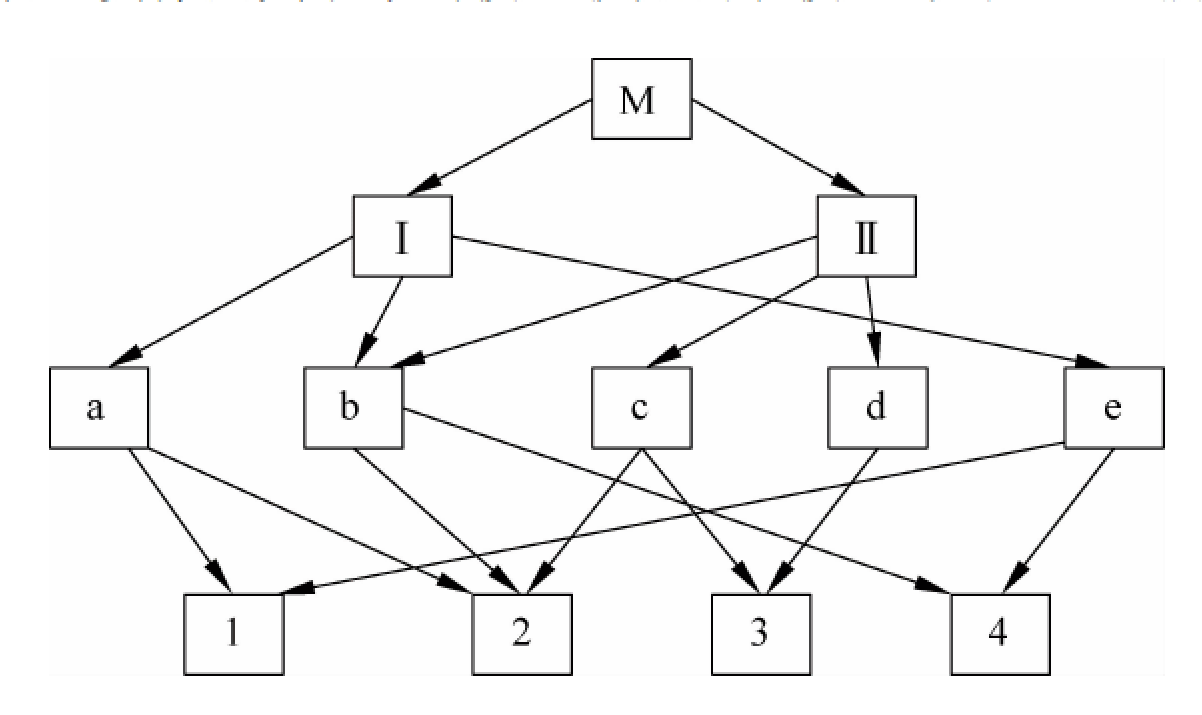


图 6-11　网状数据模型

1）网状数据模型的特点

- 可有一个以上的结点无父结点。
- 父结点与某个子结点记录之间可以有多种关系（一对一、一对多、多对多）。
- 在该模型中各记录类型间可用任意个连接关系，一个子结点可有多个父结点。

2）网状数据模型的优点

- 能够更为直接地描述现实世界，如一个结点可以有多个双亲。
- 具有良好的性能，存取效率较高。

3）网状数据模型的缺点

- 结构比较复杂，而且随着应用环境的扩大，数据库的结构就变得越来越复杂，不利于最终用户掌握。
- 数据定义语言（Data Definition Language，DDL）和数据操纵语言（Data Manipulation Language，DML）相对复杂，用户不容易使用。
- 由于数据间的联系要通过指针表示，指针数据项的存在使数据量大大增加，当数据

关系复杂时，指针部分会占用大量数据库存储空间。

- 在修改数据时，指针也必须随着变化，因此，网络数据库中的指针的建立和维护可能成为相当大的额外负担。

4）网状模型与层次模型的区别

- 网状模型允许多个结点没有双亲结点。
- 网状模型允许结点有多个双亲结点。
- 网状模型允许两个结点之间有多种联系（复合联系）。
- 网状模型可以更直接地去描述现实世界。
- 层次模型实际上是网状模型的一个特例。

3. 关系数据模型

关系数据模型是最重要的一种数据模型，也是目前主要采用的数据模型。它是以关系数学理论为基础的，用二维表结构来表示实体以及实体之间联系的模型。在关系模型中把数据看成是二维表中的元素，操作的对象和结果都是二维表，一张二维表就是一个关系。

关系模型有三个组成部分：数据结构、数据操作和完整性规则。

1）数据结构

在用户观点下，关系模型中数据的逻辑结构是一张二维表，它由行和列组成。

（1）关系数据模型的构成。

- 关系（或表）：一个关系就是一个表，如表 6-2 所示。

表 6-2　学生情况表

学号	姓名	性别	年龄	班级
110105	赵明	男	19	1
110106	张刚	男	20	1
110201	罗露	女	19	2
110306	孙斐	女	21	3
⋮	⋮	⋮	⋮	⋮

- 元组：表中的一行为一个元组（不包括表头）。
- 属性：表中的一列为一个属性。
- 主码（或关键字）：可以唯一确定一个元组和其他元组不同的属性组。
- 域：属性的取值范围。
- 分量：元组中的一个属性值。
- 关系模式：对关系的描述，一般表示为关系名（属性 1，属性 2，…，属性 n）。关系模型中没有层次模型中的链接指针，记录之间的联系是通过不同关系中的同名属性来实现的。

（2）实体及实体间的联系的表示方法。

- 实体型：直接用关系（表）表示。
- 属性：用属性名表示。
- 一对一联系：隐含在实体对应的关系中。

- 一对多联系：隐含在实体对应的关系中。
- 多对多联系：直接用关系表示。

例如：学生、课程、学生与课程之间的多对多联系。

- 学生（学号、姓名、年龄、性别、系号、年级）
- 课程（课程号、课程名、学分）
- 选修（学号、课程号、成绩）

关系必须是规范化的，并满足一定的规范条件。最基本的规范条件是关系的每一个分量必须是一个不可分的数据项。

2）数据操作

- 查询、插入、删除、更新。
- 数据操作是集合操作，操作对象和操作结果都是关系，即若干元组的集合。
- 存取路径对用户隐蔽，用户只要指出“干什么”，不必详细说明“怎么干”。

3）完整性规则

- 实体完整性
- 参照完整性
- 用户定义的完整性

4）关系数据模型和其他数据模型的区别

关系模型与层次型和网状型的本质区别在于数据描述一致、模型概念单一。在关系型数据库中，每一个关系都是一个二维表，无论实体本身还是实体间的联系均用称为“关系”的二维表来表示，它由表名、行和列组成。表的每一行代表一个元组，每一列称为一个属性。使得描述实体的数据本身能够自然地反映它们之间的联系。而传统的层次和网状模型数据库是使用链接指针来存储和体现联系的。

5）关系模型的基本特征

- 建立在关系数据理论之上，有可靠的数据基础。
- 可以描述一对一、一对多和多对多的联系。
- 表示的一致性。实体本身和实体间联系都使用关系描述。
- 关系的每个分量的不可分性，也就是不允许表中表。

6）关系模型的优点

- 建立在严格的数学概念的基础上。
- 关系模型概念清晰、结构简单，实体、实体联系和查询结果都采用关系表示，用户比较容易理解。
- 关系模型的存取路径对用户是透明的，程序员不用关心具体的存取过程，简化了程序员的工作和数据库开发建立的工作，具有较好的数据独立性和安全保密性。

7）关系模型的缺点

关系模型的缺点是，在某些实际应用中，由于存取路径对用户透明，导致其查询效率有时不如层次模型和网状模型。

为了提高查询的效率，有时需要对查询进行一些特别优化。

8）典型的关系数据库系统

- ORACLE

- SYBASE
- INFORMIX
- DB/2
- COBASE
- PBASE
- EasyBase
- DM/2
- OpenBase

6.3 数据库设计的理论与方法

6.3.1 数据库设计概述

数据库(Database)是数据管理的最新技术,具有数据结构化、最低冗余度、较高的程序与数据独立性、易于扩充、易于编制应用程序等优点。所谓数据库是指长期存储在计算机内的、有组织的、可共享的数据集合。数据库中的数据按一定的数据模型组织、描述和储存,具有较小的冗余度、较高的数据独立性和易扩展性,并可为各种用户共享。数据库的应用已经越来越广泛了,不仅大型计算机及中小型计算机,甚至微型计算机都采用先进的数据库技术来保持系统数据的整体性、完整性和共享性。

数据库设计(Database Design)是将业务对象转换为表和视图等数据库对象的过程,是数据库应用系统开发过程中首要的和基本的内容。数据库是信息系统的核心和基础,它把信息系统中的大量数据按照一定的模型组织起来,提供存储、维护、检索数据的功能,使信息系统可以方便、及时、准确地从数据库中获取所需的信息。一个信息系统的各个部分能否紧密地结合在一起以及如何结合,关键在于数据库。因此必须对数据库进行合理设计。

按照规范设计的方法,考虑数据库及其应用系统开发全过程,将数据库设计分为以下七个阶段:

(1) 规划阶段。
(2) 需求分析阶段。
(3) 概念结构设计阶段。
(4) 逻辑结构设计阶段。
(5) 物理结构设计。
(6) 数据库实施阶段。
(7) 数据库运行与维护阶段。

6.3.2 数据库规划阶段

对于数据库系统,特别是大型数据库系统或大型信息系统中的数据库群,规划阶段是十分必要的,它的好坏将直接影响到整个系统的成功与否。规划阶段一般分为以下三步。

(1) 系统调查：对客户组织进行全面调查，画出组织层次图，以了解客户的组织机构。

(2) 可行性分析：从经济、效益、法律等方面对建立数据库的可行性进行分析，然后写出可行性分析报告，组织专家讨论其可行性。

(3) 确定数据库系统的总目标和制订项目开发计划：在得到决策部门批准后，就可正式开展数据库系统的开发工作。

6.3.3 数据库需求分析

进行整个数据库设计必须准确了解与分析用户需求(包括数据和处理)。需求分析是整个设计过程的基础，是最困难、最耗费时间的一步。作为地基的需求分析是否做得充分与准确，决定了在其上构建数据库大厦的速度和大厦质量。

需求分析的任务是通过详细调查现实世界要处理的对象(组织、部门、企业等)，充分了解原系统(手工系统或计算机系统)工作概况，明确用户的各种需求，然后在此基础上确定新系统的功能。

需求分析调查的重点是“数据”和“处理”，通过调查、收集和分析，获得用户对数据库的如下需求。

(1) 信息需求：指用户需要从数据库中获得信息的内容与性质。由信息需求可以导出数据要求，即在数据库中需要存储哪些数据。

(2) 处理要求：指用户需要完成什么处理功能。明确用户对数据有什么样的处理要求，从而确定数据之间的相互关系。

(3) 安全性与完整性要求。

6.3.4 数据库概念结构设计

概念结构设计是将分析得到的用户需求抽象为概念模型的过程，即在需求分析的基础上，设计出能够满足用户需求的各种实体以及体现它们之间相互关系的概念结构设计模型。这样才能更好地、更准确地用某一数据库管理系统(Data Base Management System，DBMS)实现这些需求。它是整个数据库设计的关键。

概念结构的主要特点是能真实地、充分地反映现实世界，易于理解、易于更改，易于向关系、网状、层次等各种数据模型转换。描述概念模型的有力工具是E-R模型。

设计概念结构通常有四类方法。

(1) 自顶向下：即首先定义全局概念结构的框架，然后逐步细化。

(2) 自底向上：即首先定义各局部应用的概念结构，然后将它们集成起来，得到全局概念结构。

(3) 逐步扩张：首先定义最重要的核心概念结构，然后向外扩充，以滚雪球的方式逐步生成其他概念结构，直至总体概念结构。

(4) 混合策略：即将自顶向下和自底向上相结合，用自顶向下策略设计一个全局概念结构的框架，以它为骨架集成由自底向上策略中设计的各局部概念结构。其中最经常采用的策略是自底向上方法，即自顶向下进行需求分析，然后再自底向上设计概念结构。

6.3.5　数据库逻辑结构设计

概念结构是独立于任何一种数据模型的信息结构。逻辑结构设计的任务就是把概念结构设计阶段设计好的基本 E-R 图转换为与选用 DBMS 产品所支持的数据模型相符合的逻辑结构。

设计逻辑结构一般要分三步进行：

(1) 将概念结构转换为一般的关系模型、网状模型和层次模型。

(2) 将转换来的关系模型、网状模型和层次模型向特定 DBMS 支持下的数据模型转换。

(3) 对数据模型进行优化。

数据库的概念结构和逻辑结构设计是数据库设计过程中最重要的两个环节。

6.3.6　数据库物理结构设计

数据库在物理设备上的存储结构与存储方法称为数据库的物理结构，它依赖于给定的计算机系统。为一个给定的逻辑数据模型选择一个最适合应用要求的物理结果的过程，就是数据库的物理设计。数据库的物理结构设计通常分为两步：

(1) 确定数据库的物理结构，在关系数据库中主要指存取方法和存储结构。

(2) 对物理结构进行评价，评价的重点是时间效率和空间效率。

6.3.7　数据库实施、运行和维护

完成数据库的物理设计以后，设计人员就要用关系型数据库管理系统(Relational Database Management System，RDBMS)提供的数据定义语言和其他应用程序将数据库逻辑设计和物理设计结果严格描述出来，成为 DBMS 可以接受的源代码，再经过调试产生目标模式，然后就可以组织数据入库了。

6.4　数据模型建模实例分析

本例介绍某学校的管理信息系统。学校有四个部门要求实现计算机管理。

- 人事处：教职工管理。
- 学生处：学生学籍管理。
- 教务处：教学管理。
- 后勤处：住宅和宿舍管理。

假定在设计之前，已进行了调研和需求分析。主要信息如下所示。

- 学校包括多个管理部门和多个系。
- 每个部门或系有多名教职工，一名教职工只能属于一个部门或一个系。
- 每个系有多个班级，一个班级只能属于一个系。
- 每个班级有若干名学生，一名学生只能属于一个班级。
- 学校开设多门课程，一门课程可被多名学生选修，一名学生也可选修多门课程。

- 一门课程可由多名教师讲授,一名教师也可讲授多门课程。
- 学校有多间教室,一间教室同一时间只能安排一门课程,而一门课程在同一时间安排在多个教室(由多名教师讲授,见上)。
- 学校有多座教工住宅楼,每个住宅楼有多套住房,每套住房只能分配给一名教工,每名教职工只能分配一套住房。
- 学校有多座学生宿舍楼,每个宿舍楼有多个房间,每个房间可安排多名学生住宿,每个学生只能安排一个房间。

6.4.1 设计局部 E-R 模型

1. 确定局部应用范围

通常情况下可按系统的使用部门划分。本例中的模型被划分为四个模块。

- 人事管理——人事处
- 学生管理——学生处
- 教学管理——教务处
- 住房管理——后勤处

通常,校长需要了解整个学校的运行情况,所以还应有一个校长查询模块,提供决策指导信息。

2. 确定实体集(以"人事管理"为例)

通过调研和需求分析,人事部门需要管理教职工、部门、职称和职务,所以实体集有:教职工、部门、职称和职务。

3. 确定联系集

决定各实体集间的联系。

- 部门-教职工:1∶N
- 部门-职称:没有联系
- 部门-职务:没有联系
- 教职工-职称:N∶1
- 教职工-职务:N∶1
- 职称-职务:没有联系

根据第二步和第三步,画出初步 E-R 图,如图 6-12 所示。

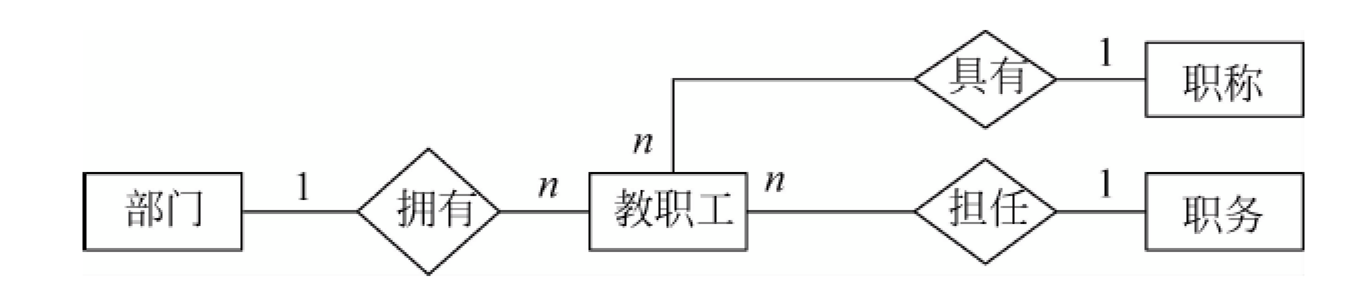


图 6-12 初步 E-R 图

4．确定实体集的属性

通过调研和需求分析，得出各实体的属性，如下所示。

- 教职工：教职工号、姓名、性别、出生日期和学历
- 部门（包括管理部门和教学院系）：部门号、类型、名称和办公电话
- 职务：代号和名称
- 职称：代号和名称

5．确定联系集的属性

- 部门-教职工：无
- 教职工-职称：聘任日期
- 教职工-职务：任职日期

6．画出局部 E-R 模型（见图 6-13）

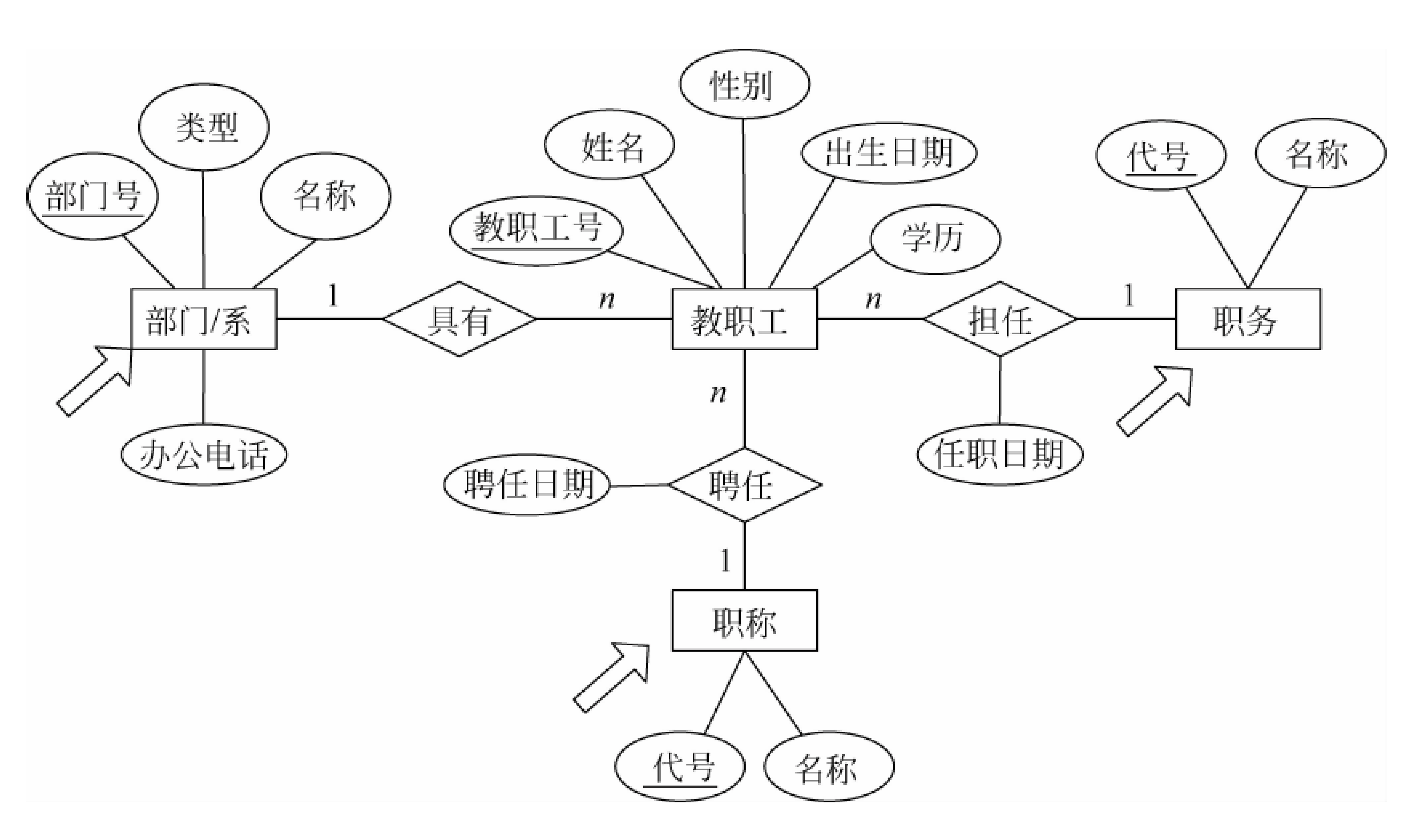


图 6-13　初步 E-R 图

7．其他三个子模块的局部 E-R 模型的设计

设计方法基本类似。

1）学生管理的局部 E-R 模型（见图 6-14）

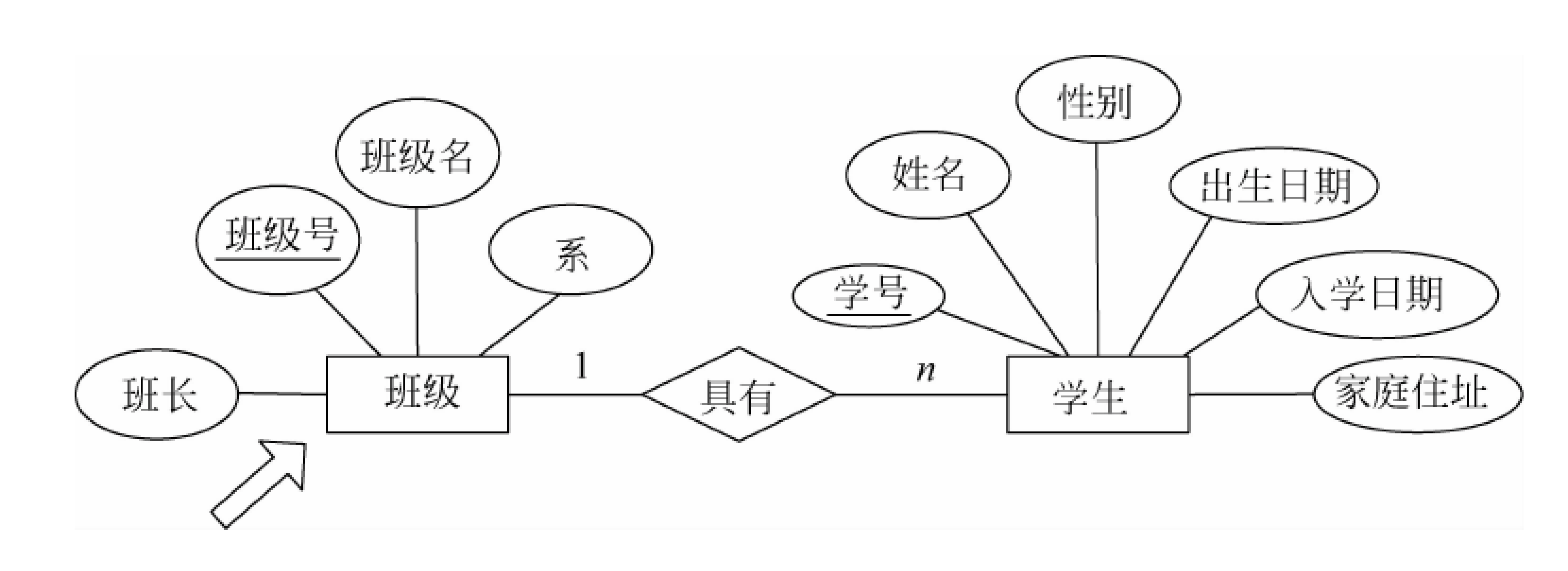


图 6-14　学生管理的局部 E-R 模型

2）教学管理的局部 E-R 模型（见图 6-15）

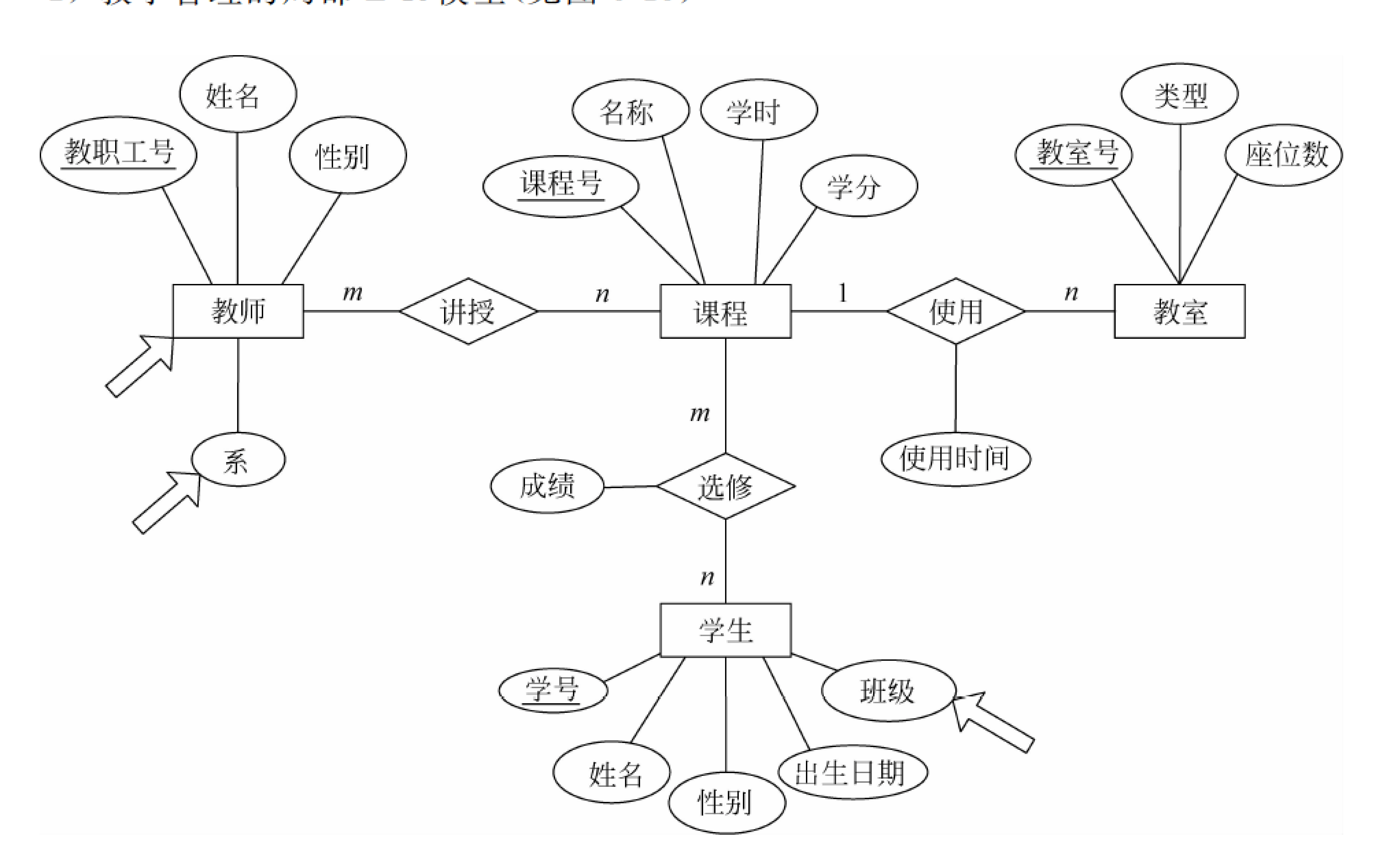


图 6-15 教学管理的局部 E-R 模型

3）住房管理的局部 E-R 模型（见图 6-16）

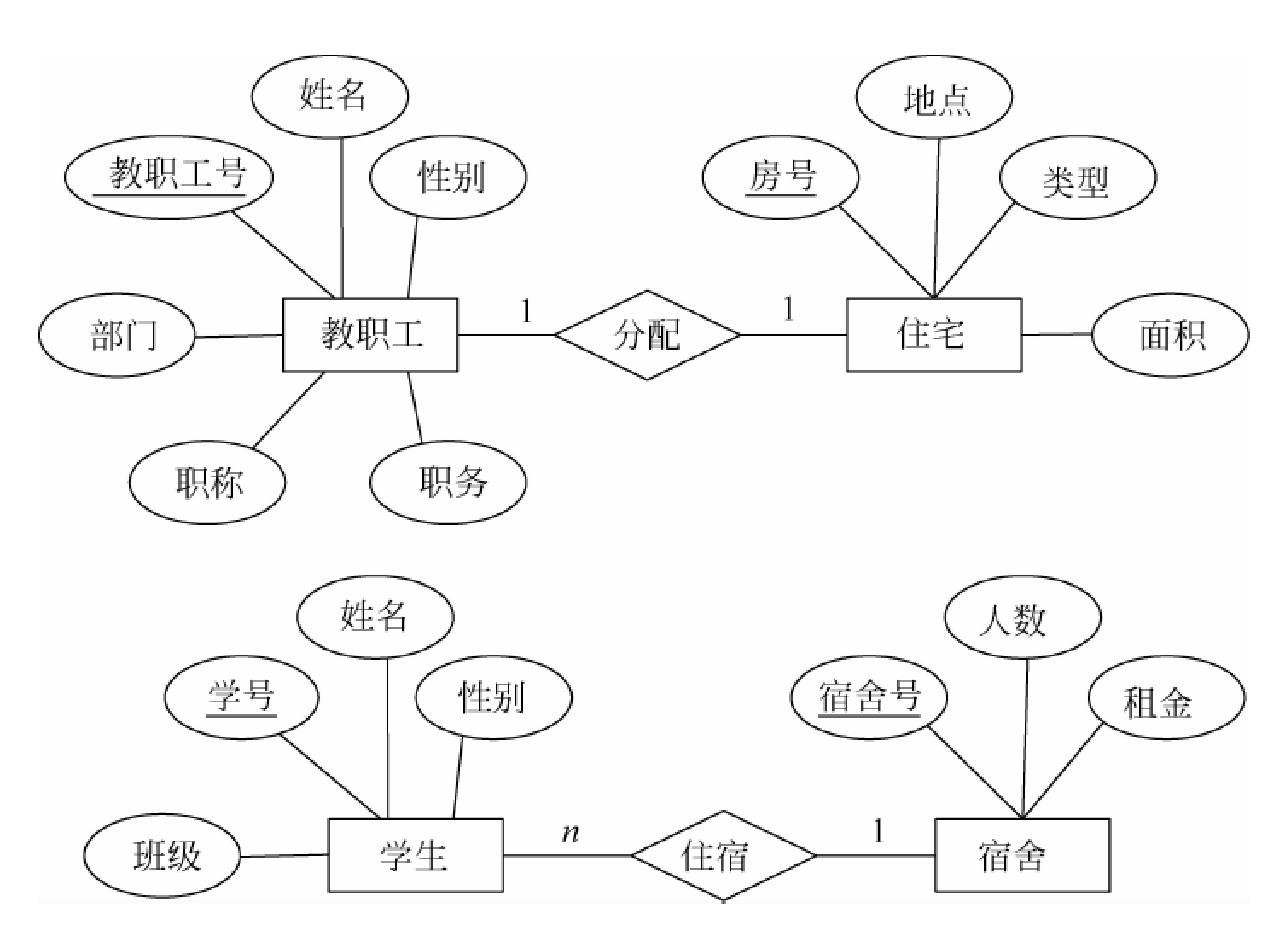


图 6-16 住房管理的局部 E-R 模型

6.4.2 设计总体 E-R 模型

（1）把所有局部 E-R 模型进行整合，形成一个统一的 E-R 模型。整合时，在兼顾各模块的需求前提下，主要应解决冲突问题。

① 命名冲突。

- 同义异名：同一对象在不同的局部 E-R 图中具有不同的名字。需要把名字进行统一。例如，将教职工和教师统一为教职工。
- 同名异义：不同的对象在不同的局部 E-R 图中具有相同的名字。需要重新命名加以区别。

② 属性冲突。

属性值类型、取值范围、取值单位（如公斤、磅）需要进行统一，以同时满足各部门的需求。

③ 结构冲突。

- 同一对象在不同的局部 E-R 图中抽象级别不同。需要进行调整。如系在教学管理中为属性，而在人事管理中为实体。
- 同一实体在不同的局部 E-R 图中包含的属性个数不同。需要进行统一。如教学管理的教师实体与人事管理的教职工实体。

(2) 在进行整合时，有时可能需要增加新的联系。

例如，整合人事管理与学生管理时，在部门（系）实体与班级实体之间可以增加一个联系：系、班级。

(3) 有时还可能要删除冗余的联系，这主要是指从其他联系可以推导出来的联系。

为便于检查联系，可把所有实体的属性临时删除。

(4) 在整合时，一般采用两两整合的方法，直到所有局部 E-R 图合并成一个完整的总体 E-R 图。

注意选择一个公共的关键实体，将它作为基准进行两两整合。本例中可择取学生实体或课程实体。

6.4.3　消除冗余、优化总体 E-R 模型

(1) 优化的目标是在满足需求的前提下：

- 实体的属性尽可能少
- 实体联系尽可能少
- 实体的属性无冗余
- 实体间的联系无冗余

(2) 在本例中：职称和职务实体只有一个有效属性“名称”，所以可以把它们作为教职工的属性，同时把相关的联系的属性（聘任日期和任职日期）也作为教职工的属性。

(3) 最终完成的总体 E-R 图（见图 6-17）。

在优化时应注意，消除冗余不是绝对的。为了提高效率，有时必要的冗余也是允许的。冗余数据的完整性（一致性）可通过两种方法解决。

- 对一个数据的增、删、改时，对另一个数据也进行同步操作，将两个操作放在一个事务中。
- 使用触发器。对一个数据的增、删、改，用触发器对另一个数据进行同步操作。

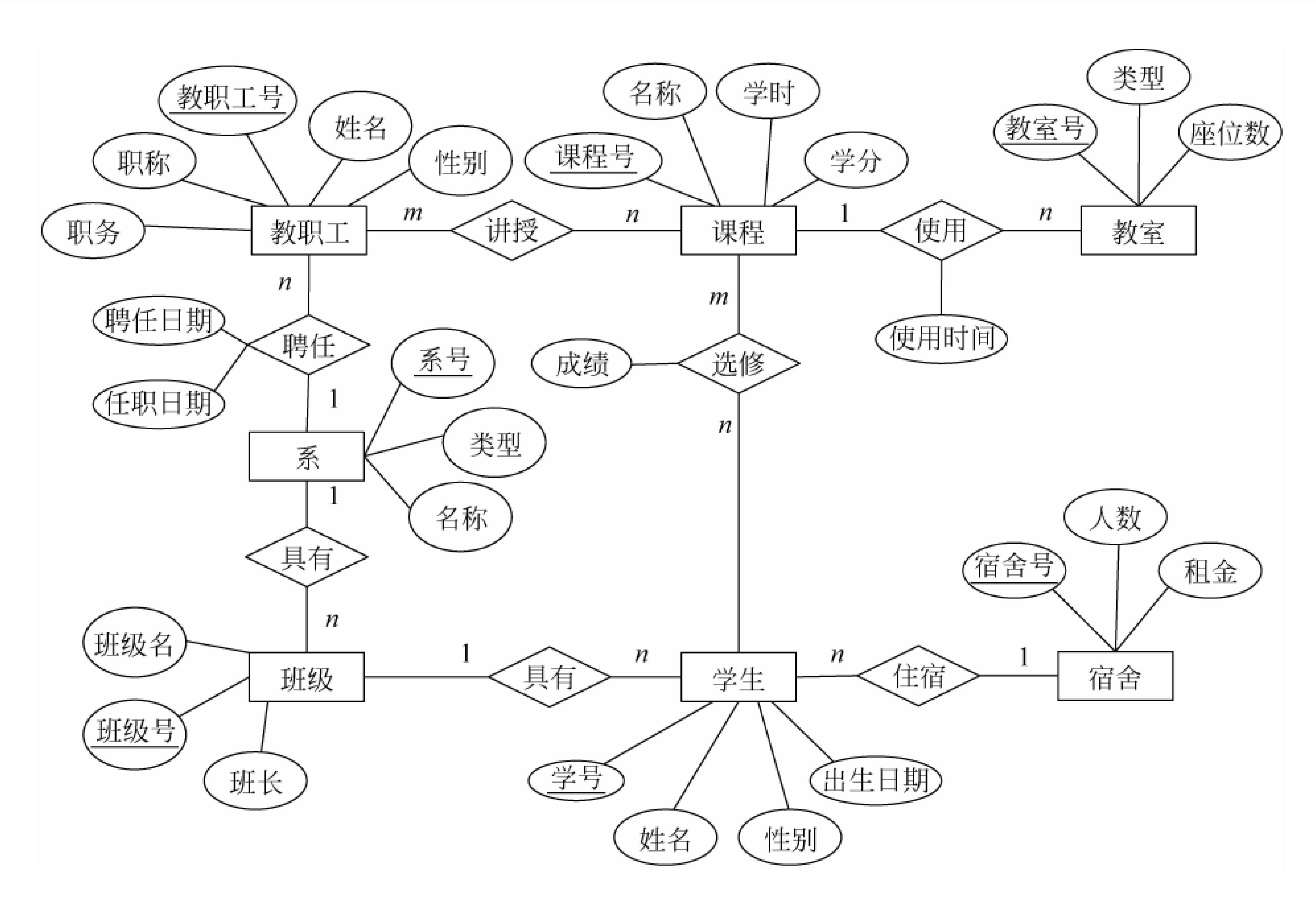


图 6-17 总体 E-R 模型

6.5 三个模型建模实例分析

6.5.1 ATM 系统需求

某银行拟开发一个自动取款机系统，它是一个由自动取款机、中央计算机、分行计算机及柜员组成的网络系统。ATM 和中央计算机由总行投资购买。总行拥有多台 ATM，分别设在各主要街道上。分行负责提供分行计算机和柜员终端。柜员终端设在分行营业厅及分行下属的各个储蓄所内。该系统的软件开发成本由各个分行分摊。

银行柜员使用柜员终端处理储户提交的储蓄事务。储户可以用现金或支票向自己拥有的某个账户内存款或开新账户。储户也可以从自己的账户中取款。通常，一个储户可能拥有多个账户。柜员负责把储户提交的存款或取款事务输入到柜员终端，接收储户交来的现金或支票，或付给储户现金。柜员终端与相应的分行计算机通信，分行计算机具体处理针对某个账户的事务并且维护账户。拥有银行账户的储户有权申请领取现金兑换卡。使用现金兑换卡可以通过 ATM 访问自己的账户。目前仅限于用现金兑换卡在 ATM 上提取现金(即取款)，或查询有关自己账户的信息(例如，某个指定账户上的余额)。

所谓现金兑换卡就是一张特制的磁卡，上面有分行代码和卡号。分行代码唯一标识总行下属的一个分行，卡号确定了这张卡可以访问那些账户。通常，一张卡可以访问储户的若干账户，但是不一定能访问这个储户的全部账户。每张现金兑换卡仅属于一个储户所有，但是，同一张卡可能有多个副本。因此，必须考虑同时在若干台 ATM 上使用同样的现金兑换卡的可能性。也就是说，系统应该能够处理并发访问。

当用户把现金兑换卡插入 ATM 之后，ATM 就与用户进行交互，以获取有关这次事务的信息，并与计算机交换关于事务的信息。首先，ATM 要求用户输入密码，接下来 ATM 把从这张卡上读取的信息以及用户输入的密码传输给中央计算机，请求中央计算机核对这些信息并处理这次事务。中央计算机根据卡上的分行代码确定这次事务与分行的对应关系，并且委托相应的分行计算机验证用户密码。如果用户输入的密码是正确的，ATM 就要求用户选择事务的类型(取款、查询等)。当用户选择取款时，ATM 请求用户输入取款额。最后，ATM 从现金出口吐出现金，并且还可为用户打印账户单。ATM 系统模型如图 6-18 所示。

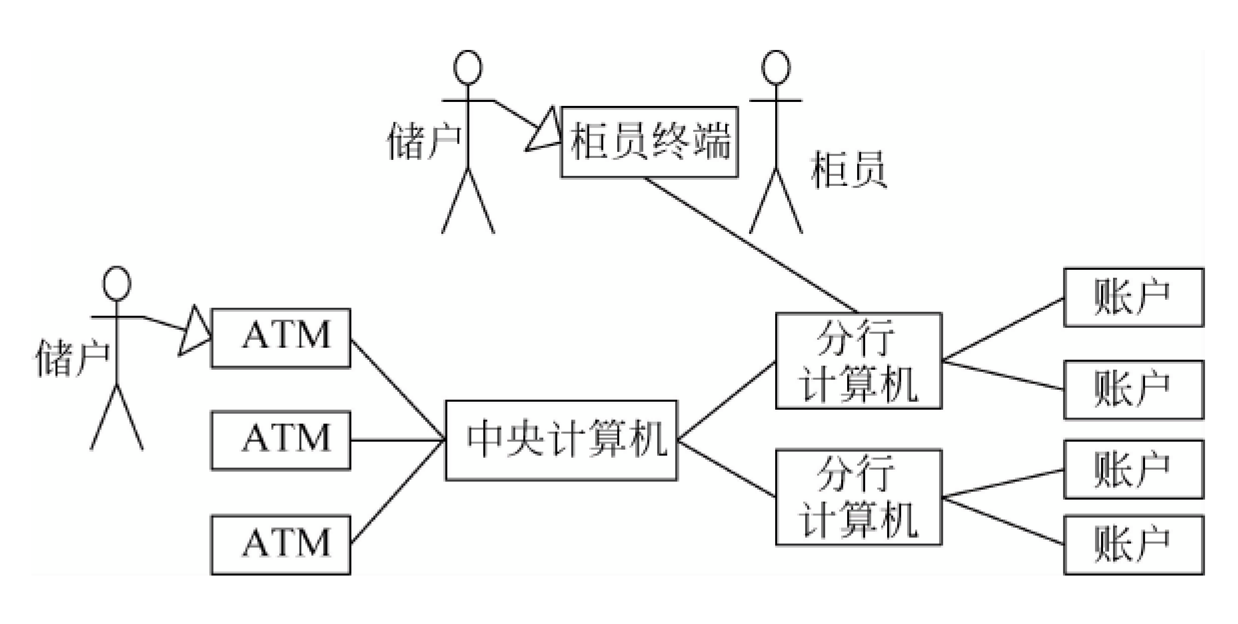


图 6-18 ATM 系统模型

6.5.2 建立对象模型

1. 确定类与对象

1) 找出候选的类与对象

通过 6.5.1 节给出的系统需求陈述中找出下列名词，可以把它们作为类与对象的初步的候选者：银行、自动取款机(ATM)、系统、中央计算机业、分行计算机、柜员终端、网络、总行、分行、软件、成本、市、街道、营业厅、储蓄所、柜员、储户、现金、支票、账户、事务、现金兑换卡、余额、磁卡、分行代码、卡号、用户、副本、信息、密码、类型、取款额、账单和访问。

在 ATM 系统的需求陈述中虽然没写“通信链路”和“事物日志”，但是，根据领域知识和常识可以知道，在 ATM 系统中应该包含这两个实体。

2) 筛选出正确的类与对象

筛选时主要依据下列标准，删除不正确或不必要的类与对象。

- 冗余。在 ATM 系统中，上面用非正式分析法得出了 34 个候选的类。其中储户与用户，现金兑换卡与磁卡及副本分别描述了相同的两类信息，因此，应该去掉“用户”、“磁卡”、“副本”等冗余的类，仅保留“储户”和“现金兑换卡”这两个类。
- 无关。ATM 系统并不处理分摊软件开发成本的问题，而且 ATM 和柜员终端放置的地点与本软件的关系也不大。因此，应该去掉候选类“成本”、“市”、“街道”、“营业厅”和“储蓄所”。
- 笼统。“银行”实际指总行或分行，“访问”在这里实际指事务，“信息”的具体内容在需求陈述中随后就指明了，所以应该去掉“银行”、“网络”、“系统”、“软件”、“信息”、

"访问"。

- 属性。在 ATM 系统中,"现金"、"支票"、"取款额"、"账单"、"余额"、"分行代码"、"卡号"、"密码"、"类型"等,都应该作为属性对待。
- 实现。"事务日志"无非是对一系列事务的记录,它的确切表示方式是面向对象设计的议题;"通信链路"在逻辑上是一种联系,在系统实现时它是关联类的物理实现。所以,应该暂时去掉"事务日志"和"通信链路"这两个类,在设计或实现时再考虑它们。

综上分析,在 ATM 系统中,经过初步筛选,剩下下列类与对象:ATM、中央计算机、分行计算机、柜员终端、总行、分行、柜员、储户、账户、事物、现金兑换卡。

2. 确定关联

1) 初步确定关联

经过分析初步确定出下列关联。

(1) 直接提取动词短语得出的关联:

- ATM、中央计算机、分行计算机及柜员终端组成网络。
- 总行拥有多台 ATM。
- ATM 设在主要街道上。
- 分行提供分行计算机和柜员终端。
- 柜员终端设在分行营业厅及储蓄所内。
- 分行分摊软件开发成本。
- 储户拥有账户。
- 分行计算机处理针对账户的事务。
- 分行计算机维护账户。
- 柜员终端与分行计算机通信。
- 柜员输入针对账户的事务。
- ATM 与中央计算机交换关于事务的信息。
- 中央计算机确定事务与分行的对应关系。
- ATM 读取现金兑换卡。
- ATM 与用户交互。
- ATM 吐出现金。
- ATM 打印账单。
- 系统处理并发访问。

(2) 需求陈述中隐含的关联:

- 总行由各个分行组成。
- 分行保管账户。
- 总行拥有中央计算机。
- 系统维护事务日志。
- 系统提供必要的安全性。
- 储户拥有现金兑换卡。

(3) 根据问题域知识得出的关联：

- 通过现金兑换卡访问账户。
- 分行雇佣柜员。

2) 筛选

筛选时主要根据下述标准删除候选的关联。

(1) 已删除的类之间的关联。

由于已经删去了“系统”、“网络”、“市”、“街道”、“成本”、“软件”、“事务日志”、“现金”、“营业厅”、“储蓄所”、“账单”等候选类，因此，与这些类有关的下列八个关联也应该被删除。

- ATM、中央计算机、分行计算机及柜员终端组成网络。
- ATM 设在主要街道上。
- 分行分摊软件开发成本。
- 系统提供必要的安全性。
- 系统维护事务日志。
- ATM 吐出现金。
- ATM 打印账单。
- 柜员终端设在分行营业厅及储蓄所内。

(2) 与问题无关的或应在实现阶段考虑的关联。

“系统处理并发的访问”并没有标明对象之间的新关联，它只不过提醒人们在实现阶段需要使用实现并发访问的算法，以处理并发事务。所以删除“系统处理并发的访问”。

(3) 瞬时事件。

“ATM 读取现金兑换卡”描述了 ATM 与用户交互周期中的一个动作，它并不是 ATM 与现金兑换卡之间的固有关系，因此应该删除。类似地，还应该删除“ATM 与用户交互”这个候选的关联。

因为“中央计算机确定事务与分行的对应关系”的动作表述隐含了结构上“中央计算机与分行通信”的关系，所以应当重新表示这个关联。

(4) 三元关联。

“柜员输入针对账户的事务”可以分解成“柜员输入事务”和“事务修改账户”这样两个二元关联。而“分行计算机处理针对账户的事务”也可以做类似分解。“ATM 与中央计算机交换关于事务的信息”这个候选的关联，实际上隐含了“ATM 与中央计算机通信”和“在 ATM 上输入事务”这两个二元关联。

(5) 派生关联。

“总行拥有多台 ATM”实质上是“总行拥有中央计算机”和“ATM 与中央计算机通信”这两个关联组合的结果。而“分行计算机维护账户”的实际含义是“分行保管账户”和“事务修改账户”。

3) 进一步改善

应该进一步完善经筛选后余下的关联，通常从下述几个方面进行改进：

(1) 正名。“分行提供分行计算机和柜员终端”应改为“分行拥有分行计算机”和“分行拥有柜员终端”。

(2) 分解。把"事务"分解成"远程事务"和"柜员事务"。

(3) 补充。把"事务"分解成上述两类之后,需要补充"柜员输入柜员事务"、"柜员事务输进柜员终端"、"在 ATM 上输入远程事务"、"远程事务有现金兑换卡授权"等关联。

(4) 标明重数。初步判定各个关联的类型,并粗略地确定关联的重数。但是,无须为此花费过多精力,因为在分析过程中随着认识的逐渐深入,重数也会经常改动。

图 6-19 是经上述分解过程之后得出的 ATM 系统原始的类图。

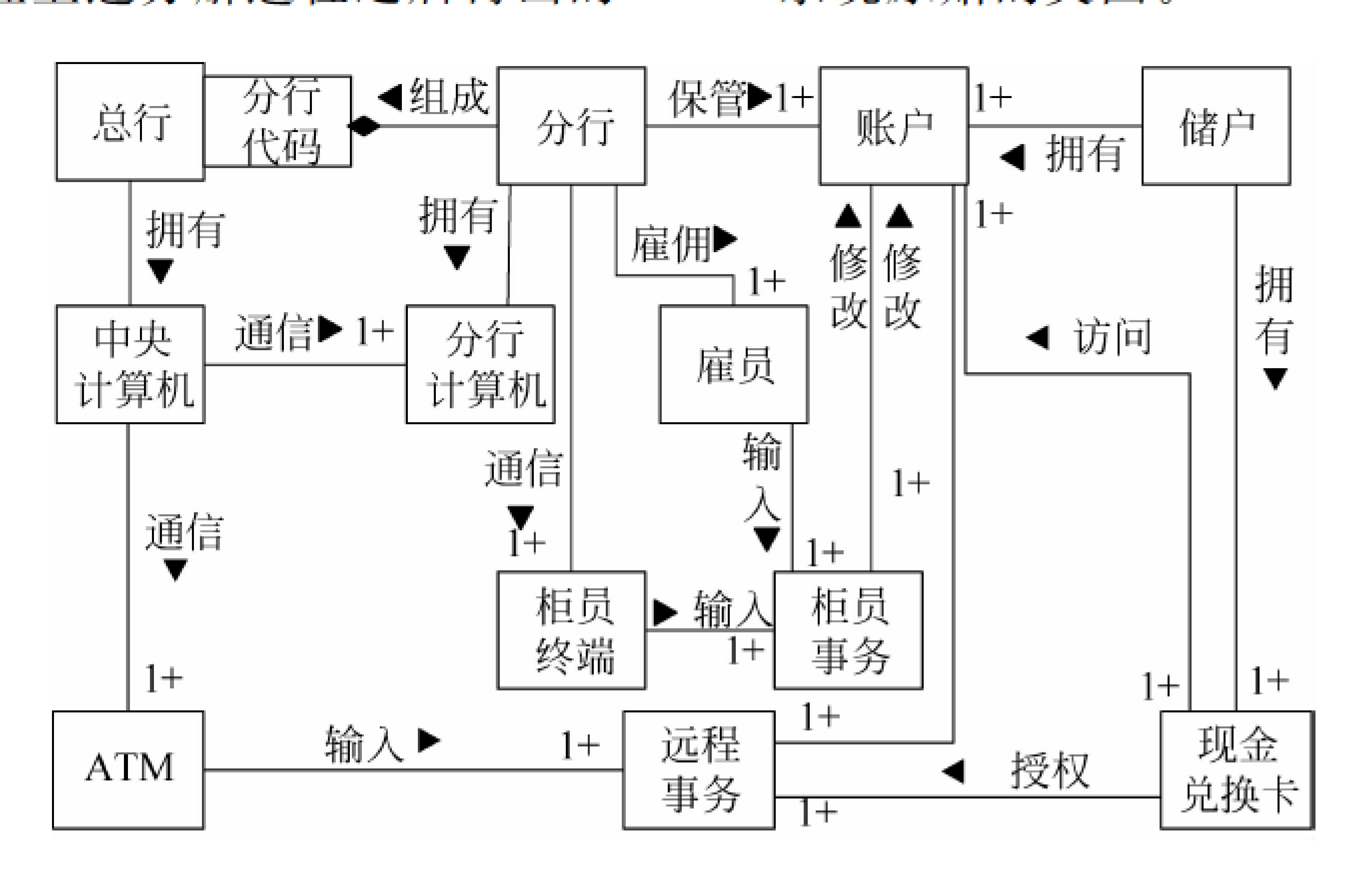


图 6-19 ATM 系统原始的类图

3. 划分主题

以 ATM 系统为例,可以把它划分成总行(包括总行和中央计算机这两个类)、分行(包括分行、分行计算机、柜员终端、柜员事务、柜员、账户等类)和 ATM(包括 ATM、远程事务、现金兑换卡、储户等类)等三个主题。

4. 确定属性

经过筛选之后,得到 ATM 系统中各个类的属性,如图 6-20 所示。图中还标出了一些限定词。

- "卡号"实际上是一个限定词。在研究卡号含义的过程中,发现以前在分析确定关联的过程中遗漏了"分行发放现金兑换卡"这个关联,现在把这个关联补上,卡号是这个关联上的限定词。
- "分行代码"是关联"分行组成总行"上的限定词。
- "账号"是关联"分行保管账户"上的限定词。
- "雇员号"是"分行雇用柜员"上的限定词。
- "站号"是"分行拥有柜员终端"、"柜员终端与分行计算机通信"及"中央计算机与 ATM 通信"三个关联上的限定词。

5. 识别继承关系

在 ATM 系统中,"远程事务"和"柜员事务"是类似的,可以泛化出父类"事务";类似地,可以从"ATM"和"柜员终端"泛化出父类"输入站"。

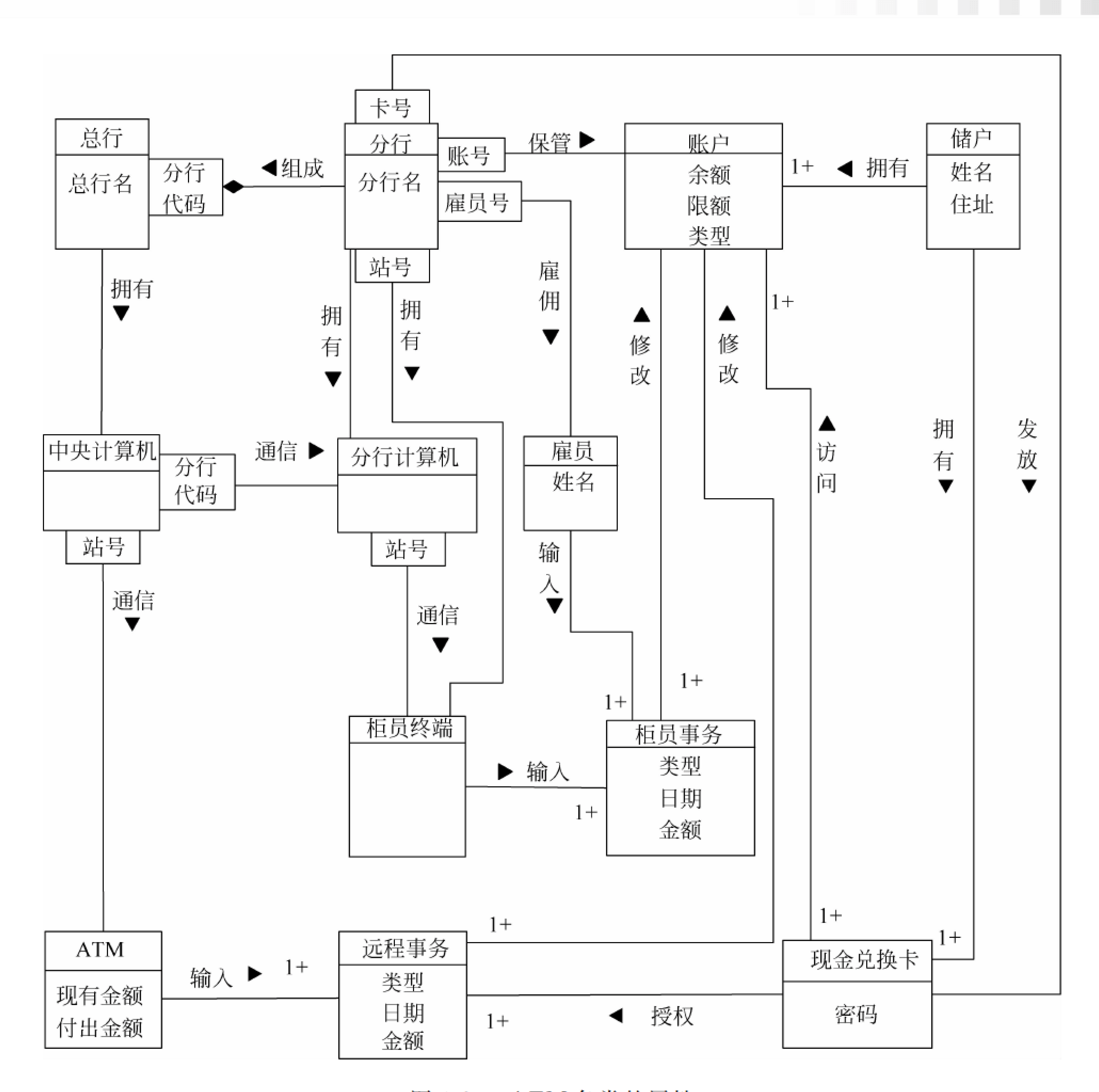


图 6-20　ATM 各类的属性

图 6-21 是增加了继承关系之后的 ATM 对象模型。

6. 反复修改

ATM 系统可能做的修改有以下几个。

1）分解“现金兑换卡”类

实际上,“现金兑换卡”有两个相对独立的功能,它既是鉴别储户使用 ATM 的权限的卡,又是 ATM 获得分行代码和卡号等数据的数据载体。因此,把“现金兑换卡”类分解为“卡权限”和“现金兑换卡”两个类,将使每个类的功能更单一:前一个类标志储户访问账户的权限,后一个类是含有分行代码和卡号的数据载体。多张现金兑换卡可能对应着相同的访问权限。

2）“事务”由“更新”组成

通常,一个事务包含对账户的若干次更新,这里所说的更新,指的是对账户所做的一个动作(取款、存款或查询)。“更新”虽然代表一个动作,但是它有自己的属性(类型、金额等),

应该独立存在，因此应该把它作为类。

3）把“分行”与“分行计算机”合并

区分“分行”与“分行计算机”，对于分析这个系统来说，并没有多大意义，为简单起见，应该把它们合并。类似地，应该合并“总行”和“中央计算机”。

图 6-22 给出了修改后的 ATM 对象模型，与修改前比较起来，它更简单、更清晰。

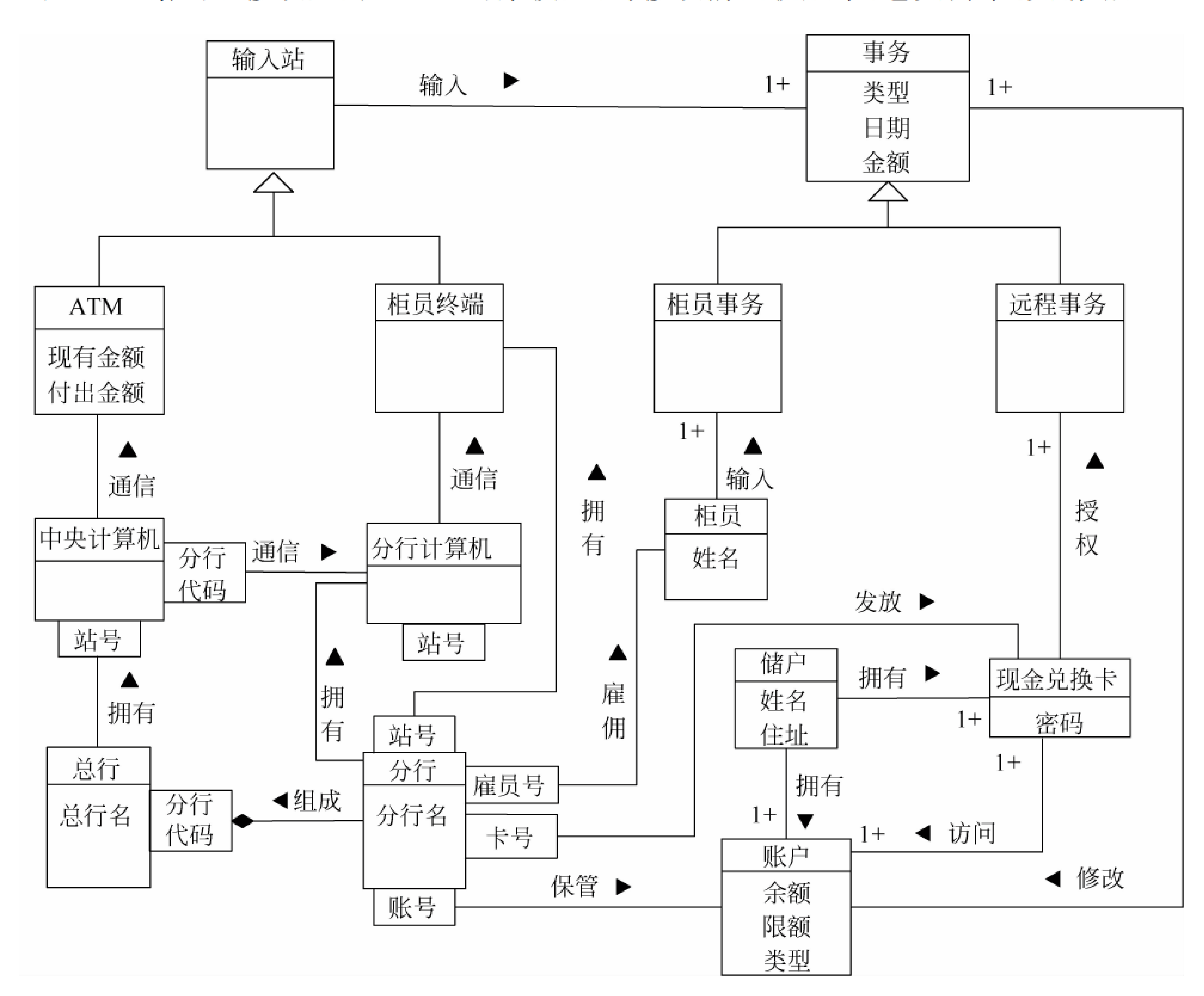


图 6-21　带继承关系的 ATM 对象模型

6.5.3　建立动态模型

1. 编写脚本

下面分别给出了 ATM 系统的正常情况脚本和异常情况脚本。

ATM 系统的正常情况脚本：

- ATM 请储户插卡；储户插入一张现金兑换卡。
- ATM 接受该卡并读取它上面的分行代码和卡号。
- ATM 要求储户输入密码；储户输入自己的密码 1234 等数字。
- ATM 请求总行验证卡号和密码；总行要求 39 号分行核对储户密码，然后通知 ATM 这张卡有效。
- ATM 要求储户选择事务类型(取款、转账、查询等)；储户选择“取款”。

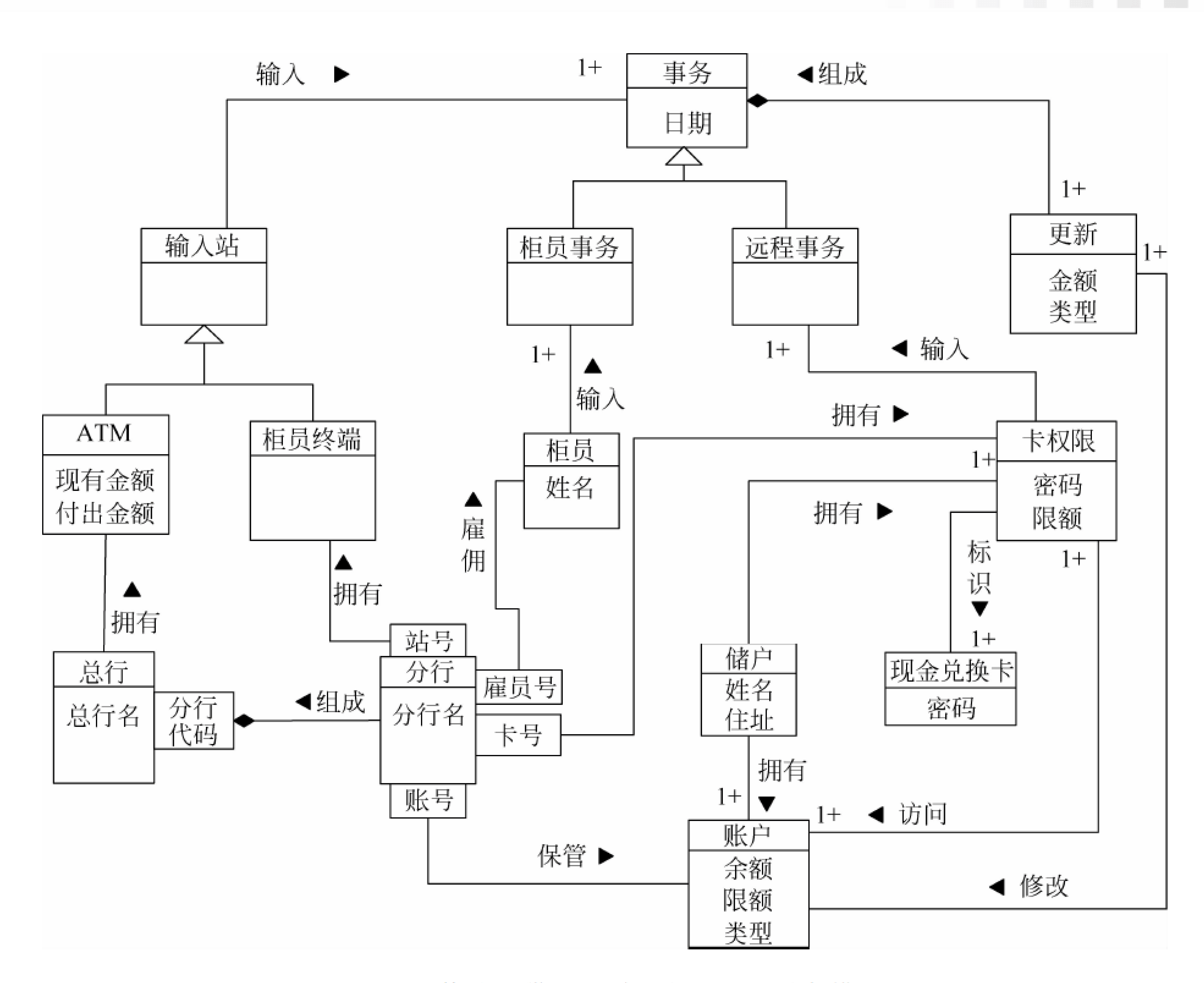


图 6-22 修改后带继承关系的 ATM 对象模型

- ATM 要求储户输入取款额；储户输入 880。
- ATM 确认取款额在预先设定的限额内，然后要求总行处理这个事务；总行把请求转给分行，该分行成功地处理完这项事务并返回该账户的新余额。
- ATM 吐出现金并请储户拿走这些现金；储户拿走现金。
- ATM 问储户是否继续执行这项事务；储户回答"不"。
- ATM 打印账单，退出现金兑换卡，请储户拿走它们；储户取走账单和卡。
- ATM 请储户插卡。

ATM 系统的异常情况脚本：

- ATM 请储户插卡；储户插入一张现金兑换卡。
- ATM 接受该卡并读取它上面的分行代码和卡号。
- ATM 要求储户输入密码；储户误输入 8888。
- ATM 请求总行验证输入的数字和密码；总行在向有关分行咨询之后拒绝这张卡。
- ATM 显示"密码错"，并请储户重新输入密码；储户输入 1234；ATM 请总行验证后知道这次输入的密码正确。
- ATM 要求储户选择事务类型；储户选择"取款"。
- ATM 询问取款额；储户改变主意不想取款了，他按下"取消"键。
- ATM 退出现金兑换卡，请储户取走卡；储户取走卡。
- ATM 请储户插卡。

2. 设想用户界面(见图 6-23)

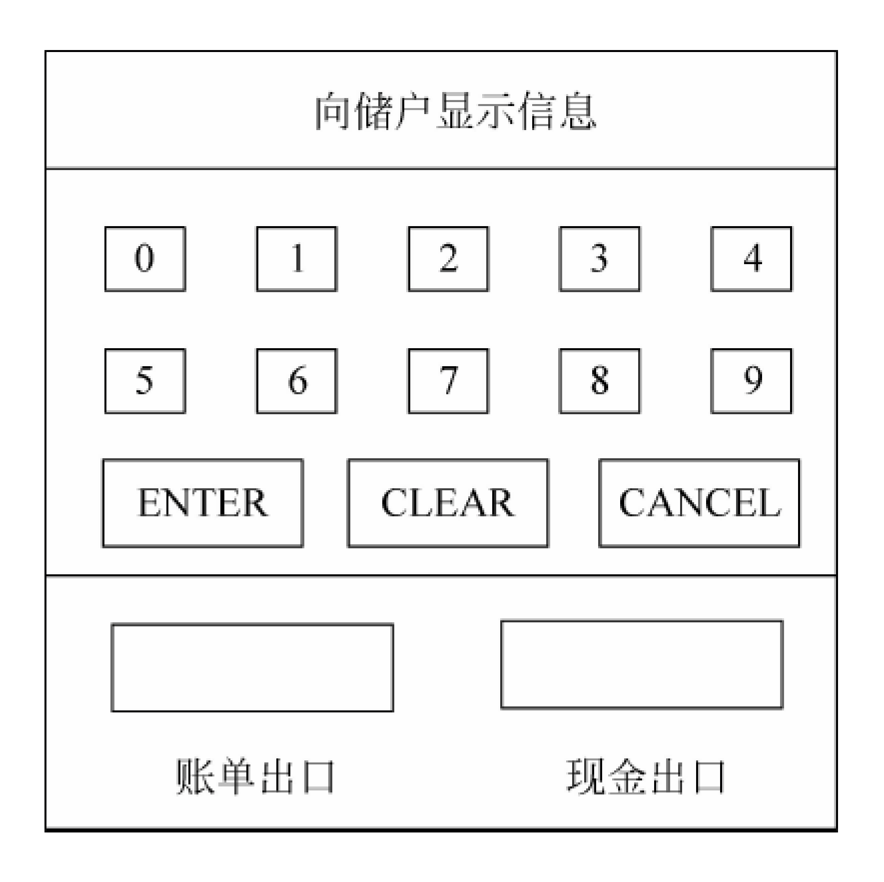


图 6-23 ATM 的界面格式

3. 画事件跟踪图

图 6-24 是 ATM 系统正常情况下的事件跟踪图。

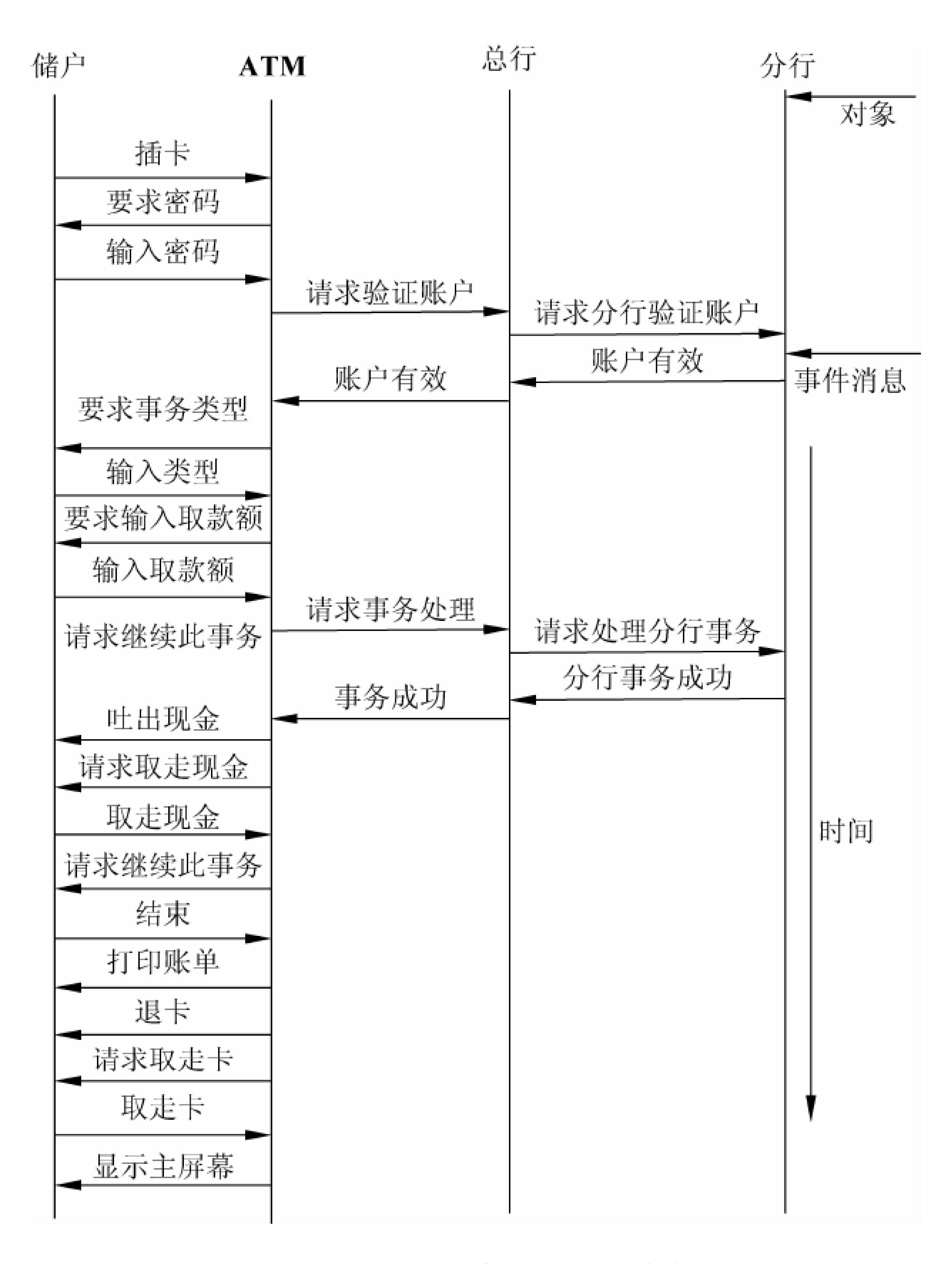


图 6-24 ATM 系统正常情况下的事件跟踪图

4．画状态图

“ATM”、“柜员终端”、“总行”和“分行”都是主动对象，它们相互发送事件；而“现金兑换卡”、“事物”、“账户”是被动对象，并不发送事件。“储户”和“柜员”虽然也是动作对象，但是，它们都是系统外部的因素，无须在系统内实现它们。因此，只需要考虑“ATM”、“总行”、“柜员终端”和“分行”的状态图。图 6-25、图 6-26 和图 6-27 分别是“ATM”、“总行”和“分行”的状态图。

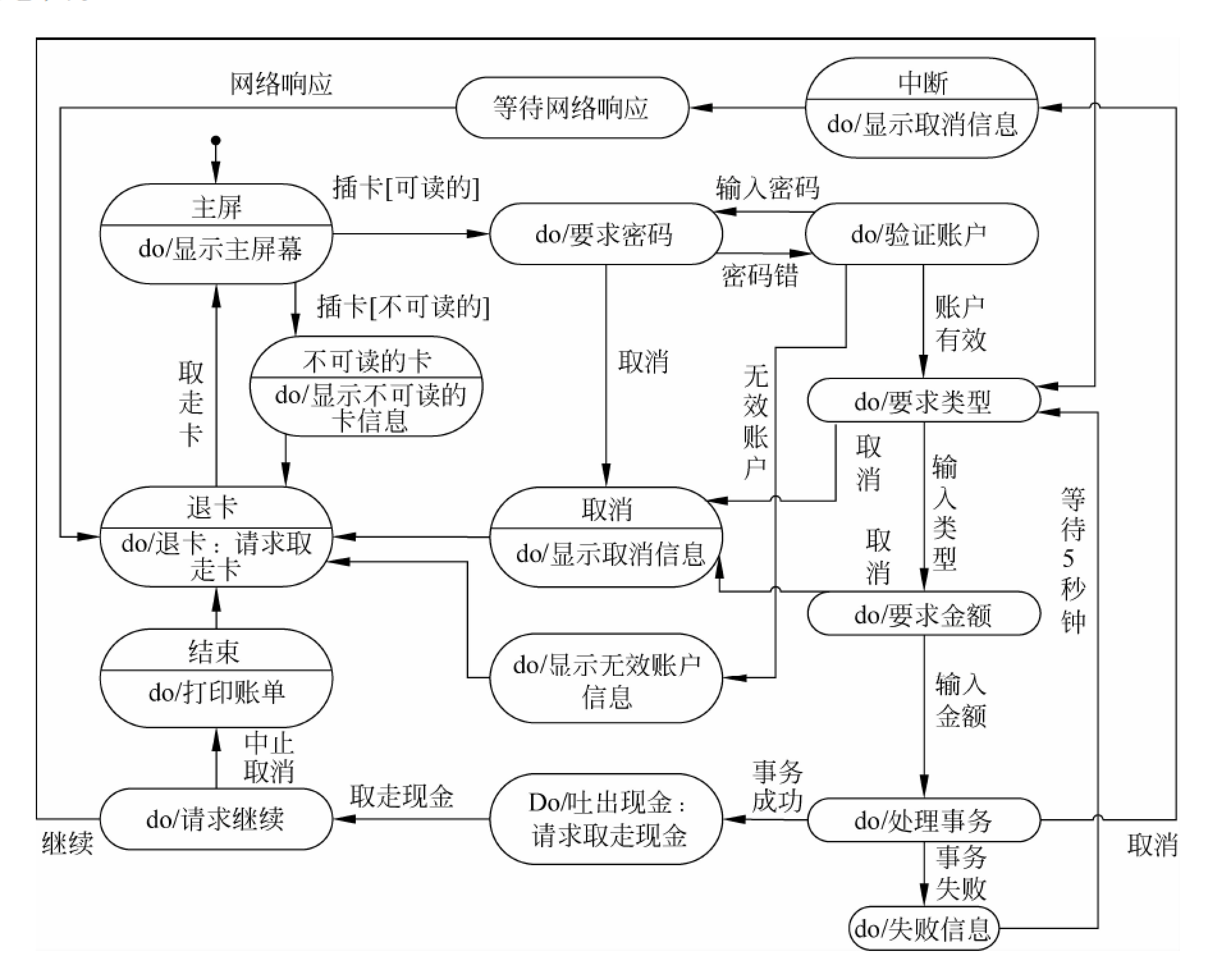


图 6-25　ATM 的状态图

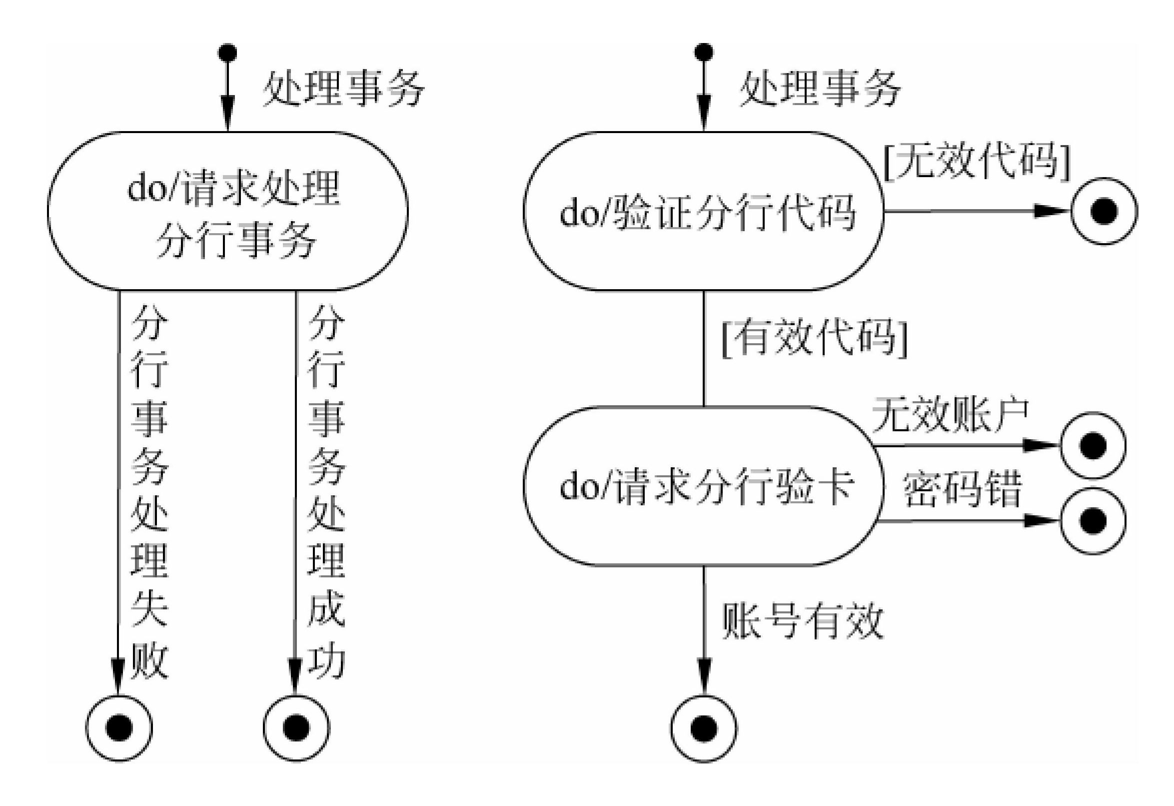


图 6-26　总行类的状态图

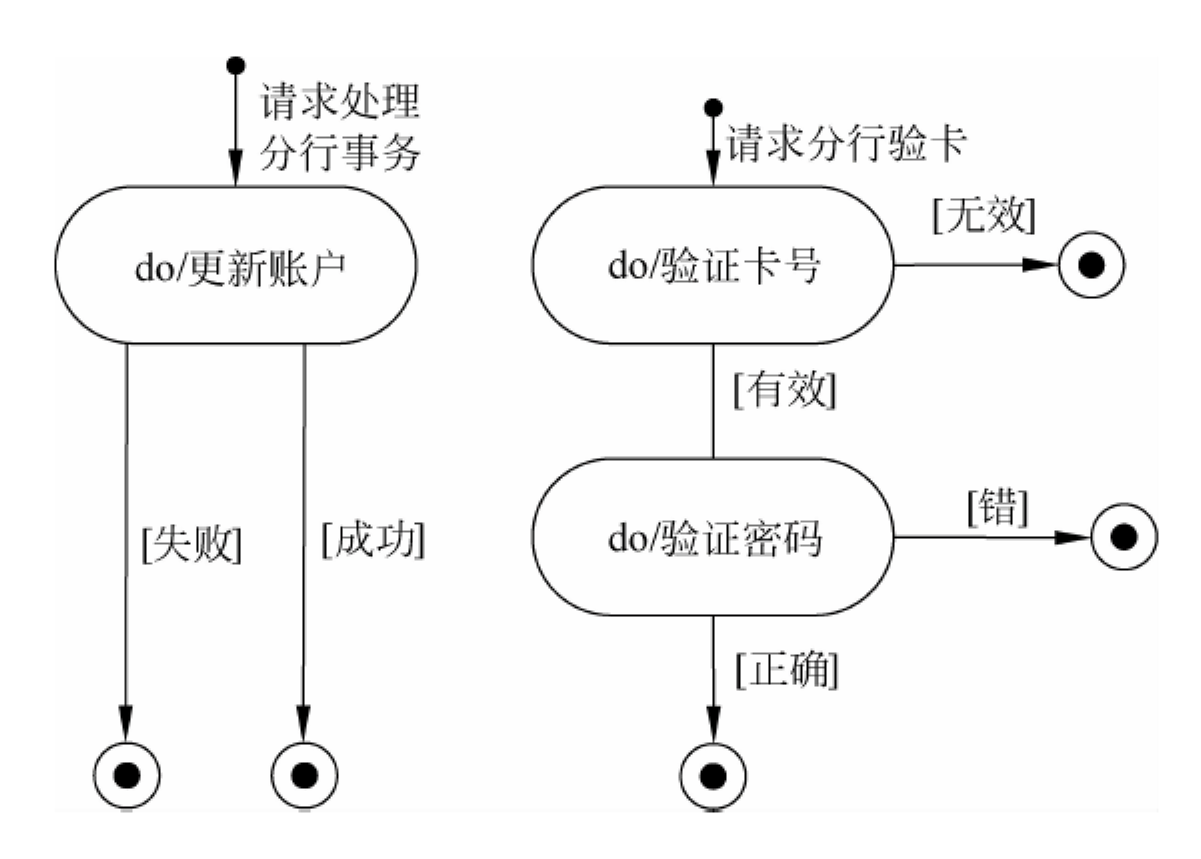


图 6-27　分行类的状态图

5. 审查动态模型

在总行类的状态图中，事件"无效代码"是由总行发出的，但是在 ATM 类的状态图中并没有在一个状态接受这个事件。因此，在 ATM 类的状态图中应该再补充一个状态"do/显示分行代码错信息"，它接受由前驱状态"do/验证账户"发出的事件"分行代码错"，它的后续状态是"退卡"。

6.5.4　建立功能模型

1. 画出基本系统模型图

图 6-28 是 ATM 系统的基本系统模型。尽管在储蓄所内储户的事务是由柜员通过柜员终端提交给系统的，但是信息的来源和最终接受者都是储户，因此，本系统的数据源点或终点为储户。另一个数据源点是现金兑换卡，因为系统从它上面读取分行代码和卡号等信息。

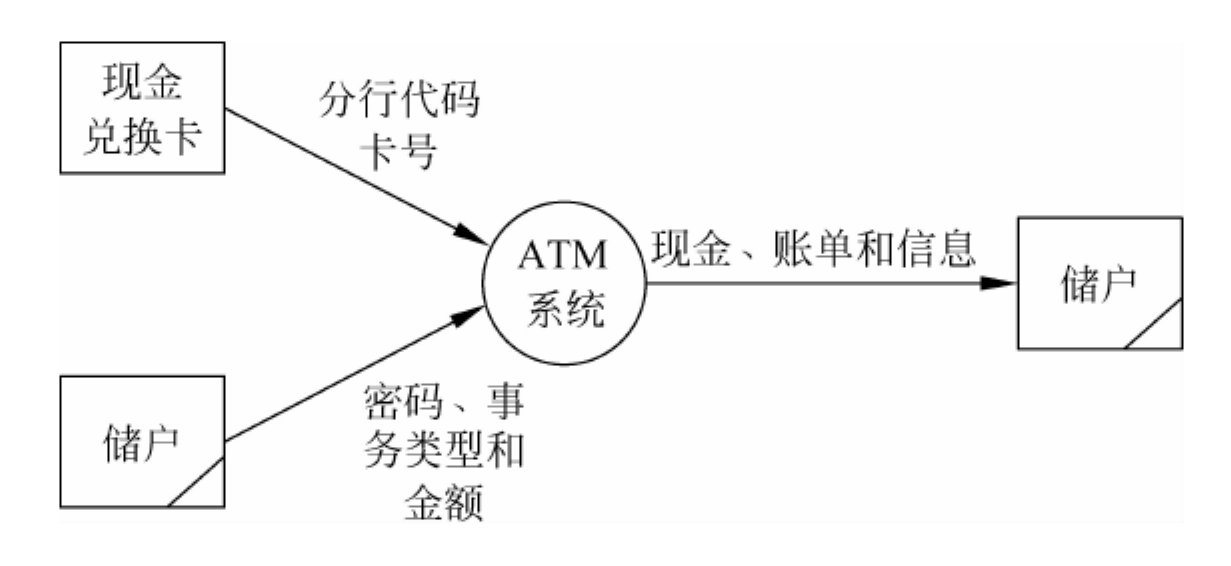


图 6-28　ATM 系统的基本系统模型

2. 画出功能级数据流图

ATM 系统的功能级数据流图如图 6-29 所示。

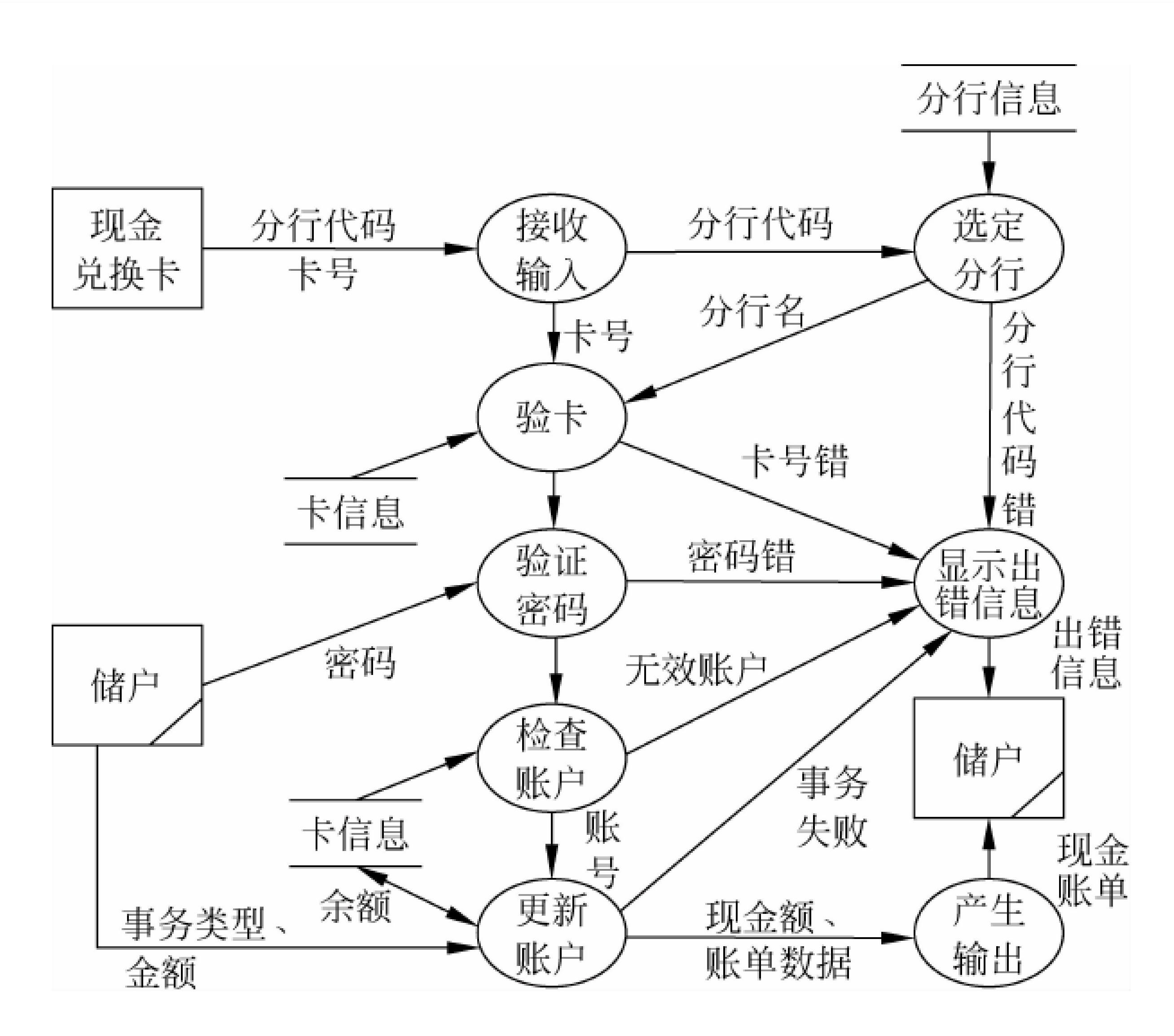


图 6-29 ATM 系统的功能级数据流图

3. 描述处理框功能

下面给出了对“更新账户”这个处理功能的描述。

- 更新账户(账号、事务类型、金额)→现金额、账单数据、信息。
- 如果取款额超过账户当前余额,则拒绝该事务且不付出现金。
- 如果取款额不超过账户当前余额,从余额中减去取款额后作为新的余额,付出储户要取的现金。
- 如果事务是存款,把存款额加到余额中得到新的余额,不付出现金。
- 如果事务是查询,不付出现金。
- 在上述任何一种情况下,账单的内容都是:ATM 号、日期、时间、账号、事务类型、事务金额(如果有的话)和新余额。

6.6 三个模型建模思想的总结

三个模型既是一种软件建模思想,又是一种建模方法,它不但告诉人们应该在什么时候、用什么方法、去建立什么模型,而且告诉人们这三个模型之间的关系,以及如何用这三个模型去解决实际问题。

6.6.1 三个模型建模思想的优点

三个模型的建模思想明显具有下列优点。

(1) 符合中国人的心理。

中国人在软件开发过程中,十多年来已经形成了一套具有中国特色的做法:

- 系统的数据怎样组织和维护:对应系统的“对象模型”。

• 系统怎么操作：对应系统的“动态模型”。
• 系统有什么功能：对应系统的“功能模型”。

(2) 符合客观事物的发展规律。

因为做任何事情，都必须回答三个问题：

• 在什么地方做？做事的原材料在什么地方？做完后的产品放到什么地方？是系统“对象模型”的任务。
• 怎么做？做到什么程度？是系统“动态模型”的任务。
• 做什么？是系统“功能模型”的任务。

(3) 符合将复杂问题简单化和抓主要矛盾的哲学思想。

项目经理的主要精力是“三抓”：

• 抓系统的“对象模型”。
• 抓系统的“动态模型”。
• 抓系统的“功能模型”。

(4) 符合“简单、方便和直观”的原则。

因为软件工程是一门工程科学、一种实用技术。

• “对象模型”听得懂：实体、属性、关系、表、字段、记录、数据字典、原始数据、统计数据和临时数据。
• “动态模型”摸得着：操作说明书和状态图。
• “功能模型”看得见：菜单、界面和报表。

(5) 符合节省成本降低费用的经济效益目标。

中国软件的开发方法与文档标准，不需要与发达国家完全相同，而应结合中国的国情来制定。

(6) 三个模型的建模思想与建模方法，对面向过程方法建模、面向对象方法建模、面向元数据方法建模都适合，对应用软件建模和系统软件建模也适合。

(7) 三个模型从根本上满足了 B/A/S。

B 层(又称浏览层)对应功能模型；A 层(又称业务逻辑层)对应动态模型；S 层(又称数据库服务器层)对应对象模型。

6.6.2 三个模型建模思想的缺点

三个模型建模思想具有以下缺点：

(1) 虽然三个模型对软件实现、软件测试这两个阶段具有重要的指导意义，但是三个模型的建模目前只能覆盖需求分析和设计两个阶段，不能覆盖整个软件生命周期。对象模型主要适合在软件设计阶段建模，动态模型和功能模型主要适合在软件需求阶段建模。

(2) 动态模型表述不规范。

(3) 功能模型表述不规范。

6.6.3 值得思考的问题

下面是几个值得思考的问题：

（1）三个模型不完全是并列关系，而应以对象模型为中心，以动态模型和功能模型为两个基本点。

（2）对象模型能否可以用类图来创建？

（3）动态模型和功能模型描述方法的改进和提高，是三个模型建模思想的发展方向。

（4）“三个模型”的软件工程建模思想，加上“四种开发方法”（面向过程的方法、面向对象的方法、面向数据的方法和形式化方法）的软件工程方法论，以及“五个面向”（面向流程分析、面向数据设计、面向对象实现、面向功能测试和面向过程管理）的软件工程实践论，就构成了一个完整的方法论。

6.7　本章小结

面向对象建模技术所建立的三个模型，分别从不同侧面描述了所要开发的系统。这三个模型相互补充、相互配合，使得人们对系统的认识更加全面：功能模型指明了系统应该“做什么”；动态模型明确规定了什么时候（即在何种状态下接受了什么事件的触发）做；对象模型则定义了做事情的实体。三个模型既是一种软件建模思想，又是一种建模方法，它不但告诉人们应该在什么时候、用什么方法、去建立什么模型，而且还会告诉人们这三个模型之间的关系，以及如何用这三个模型去解决实际问题。

数据模型是对现实世界中各种事物或实体特征的数字化模拟和抽象，可分为概念数据模型、逻辑数据模型和物理数据模型三类。其三要素是数据结构、数据操作和数据约束。

概念模型最常用的图示化描述方法是“实体-联系方法”，简称 E-R 方法，用这种方法建立的概念模型被称为 E-R 模型。逻辑数据模型可分为层次数据模型、网状数据模型和关系数据模型。

数据库设计是将业务对象转换为表和视图等数据库对象的过程，分为以下七个阶段：规划阶段、需求分析阶段、概念结构设计阶段、逻辑结构设计阶段、物理结构设计阶段、数据库实施和数据库运行与维护阶段。

习题 6

1. 简述什么是对象模型、动态模型和功能模型。
2. 简述三个模型之间的关系。
3. 数据模型可分为哪三类？
4. E-R 模型有哪些特点？
5. 简述层次模型的优缺点。
6. 网状模型与层次模型的区别是什么？
7. 简述关系数据模型的三个组成部分。
8. 简述数据库设计的七个阶段。
9. 简述三个模型建模思想的优缺点。

第7章 软件设计

在需求分析阶段，确定了系统必须做什么之后，在软件的设计阶段，就要决定怎么做。软件设计的主要任务是设计软件的结构，确定软件中的每项功能都是由哪些模块组成的，掌握模块间的相互关系，以及具体的实现方法。

对任意的工程产品或系统而言，开发阶段的第一步是确定将来所要构建的制造原型或实体表现的目标构思，这个步骤是由多方面的直觉与判断力来共同决定的。这些方面包括构建类似模型的经验、一组引领模型发展的原则、一套启动质量评价的标准以及重复修改直至设计最后定型的过程本身。计算机软件设计与其他工程学科相比还处在幼年时期，仍在不断变化中，例如更新的方法、更好的算法分析以及理解力的显著进化。软件设计方法论的出现也只有三十多年，仍然缺乏深度、适应性和定量性质，通常更多地与经典工程设计学科相联系。软件设计是一种在设计者计划中通过诸如软件如何满足客户的需要、如何才能容易地实现和如何才能方便地扩展功能以适应新的需求等不同考虑的创造性活动。软件设计有很多设计方法或技巧，可通过借鉴他人的经验让这件事情完成得更好。同时，设计者们也可以利用成熟的标记法将他们的想法和计划传达给开发者以及其他相关人员，使他们更好地了解这个系统。

软件设计必须依据对软件的需求来进行，根据需求得到软件表示的过程。最初这种表示只是描绘出可直接反映功能、数据、行为需求软件的总体框架，然后进一步细化，在此框架中填入细节，把它加工成在程序细节上非常接近于源程序的软件表示，从而在编码阶段可以把这个精确表示直接翻译成用某种程序设计语言编写的程序。软件设计的结果基本上决定了最终程序代码的质量，它是开发阶段中最重要的步骤，是保证软件开发质量的关键一步。

7.1 软件设计概论

软件设计的任务是从软件需求规格说明书出发，根据需求分析阶段确定的功能设计软件系统的整体结构、划分功能模块、确定每个模块的实现算法以及编写具体的代码，形成软件的具体设计方案。

在软件分析阶段，通过分析建模，得到数据模型、功能模型和行为模型。这些模型将被传送给软件设计者，以进行数据设计、体系结构设计、接口设计和过程设计。

1. 数据设计

数据设计把分析阶段创建的信息域模型转换成软件所需要的数据结构。实体-关系图

中定义的数据对象和关系以及数据字典中给出的详细的数据内容，可以很好地为数据设计服务。部分数据设计可能和软件体系结构的设计同时进行，更详细的数据设计则可能在设计每个构建时进行。

2. 体系结构设计

体系结构设计定义了程序各模块之间的关系。它可以从系统规约、分析模型和子系统导出。

3. 接口设计

接口设计描述了软件内部、软件和协作系统之间，以及软件与用户之间如何通信。数据流图和控制流图一起提供了接口设计所需要的信息。

4. 过程设计

过程设计把系统体系结构中的结构元素转换成软件结构的过程性描述。它从处理规格说明、控制规格说明及状态转换图中获得信息。

在软件设计阶段做出的各种决策将会直接影响软件的质量，没有好的设计，就没有好的系统。

7.2 软件设计原理

对软件进行设计的过程要考虑的因素有：开发类似系统中得到的经验、指导模型演化的原理和启发规则、判定软件质量的标准以及导出最终设计表示的迭代过程。

本节将讲述在软件设计过程中应该遵循的基本原理和相关概念。

7.2.1 模块化

模块化是指解决一个复杂问题时，自顶向下逐层把系统划分成若干模块的过程，有多种属性，分别反映其内部特性。模块化是一种将复杂系统分解为更好的可管理模块的处理方式。模块化用来分割、组织和打包软件，每个模块完成一个特定的子功能，所有的模块按某种方法组装起来，成为一个整体，完成整个系统所要求的功能。例如子程序、过程、函数、宏等都是模块，又如学生信息管理系统中的学籍管理子程序是一个模块，学生信息汇总过程是一个模块，C 语言编写的某函数也是一个模块。

模块具有以下几种基本属性：接口、功能、逻辑和状态。功能、状态与接口反映模块的外部特性，逻辑反映它的内部特性。在系统的结构中，模块是可组合、分解和更换的单元。

如果一个大型程序仅仅由一个模块组成，由于它引用跨度广、变量数目多，总体复杂度大，将很难让人理解。下面的实例可以说明这一点。

设函数 $C(x)$ 定义为问题 x 的复杂程度，函数 $E(x)$ 确定解决问题 x 需要的工作量，对于两个问题 P_1 和 P_2，如果：

$$C(P_1) > C(P_2) \tag{7-1}$$

则有：

$$E(P_1) > E(P_2) \tag{7-2}$$

根据人们解决一般问题的经验，另一个规律是：

$$C(P_1 + P_2) > C(P_1) + C(P_2) \tag{7-3}$$

式(7-3)意味着一个问题由 P_1 和 P_2 两个问题组合而成，那么它的复杂程度大于分别考虑每个问题时的复杂程度之和。由式(7-1)、式(7-2)、式(7-3)得到下面的不等式：

$$E(P_1 + P_2) > E(P_1) + E(P_2) \tag{7-4}$$

式(7-4)引出了“分而治之”的结论：把复杂的问题分解成许多容易解决的小问题，原来的问题也就容易解决了。它事实上就是模块化的依据。根据式(7-4)可以得出模块在理论上可以不断细分，就好比一台计算机可以看成由运算器、控制器、存储器、输入设备和输出设备组成的，存储器又可以分解成内存和外存，外存又可以再分解成更细的对象，一直细分下去甚至可以出现分子和原子的模块。

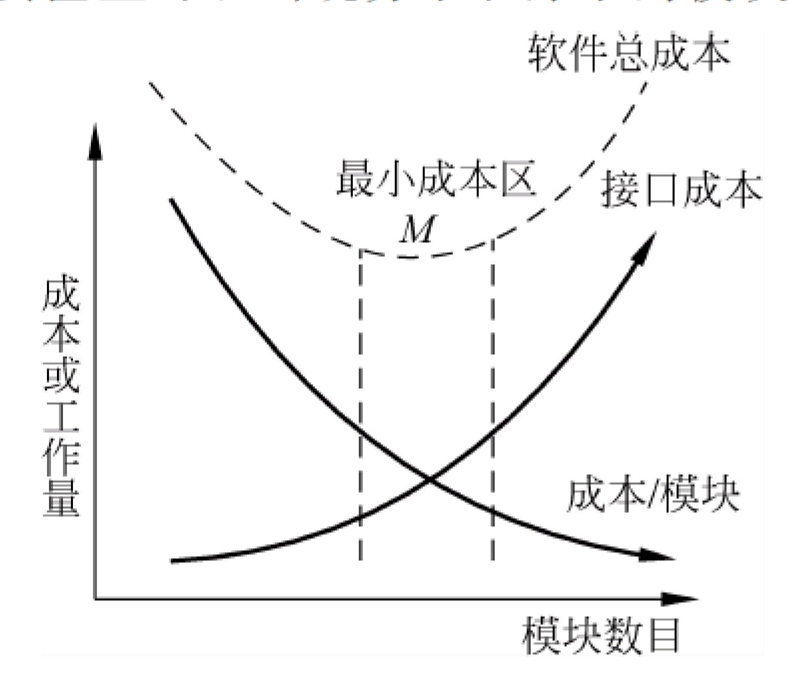


图 7-1 模块化和软件成本

如果无限制地细分模块，最终每个细小模块的工作量是否可以小到被忽略？答案是否定的，因为还有另一个因素在起作用——模块间接口的工作量。它们之间的关系如图 7-1 所示。

随着模块数的增加，每个模块的规模将减小，开发单个模块需要的成本确实减少了。但是，设计模块间接口所需要的工作量将增加，根据这两个因素，得出了图 7-1 中的总成本曲线。每个程序都相应地有一个最适当的模块数目 M，使得系统的开发成本最小。

模块评价的标准如下所示。

- 模块的可分解性：把问题分解为子问题的系统化机制。
- 模块的可组装性：把现有的可重用模块组装成新系统。
- 模块的可理解性：一个模块作为独立单元，不需要参考其他模块来理解。
- 模块的连续性：系统需求的微小修改只导致对个别模块，而不是对整个系统的修改。
- 模块的保护性：当一个模块内出现异常情况时，它的影响局限在该模块内部。

采用模块化的原理设计软件，可以使软件结构清晰，既容易设计也容易阅读和修改。程序的错误一般容易出现在模块之间的接口中，模块化使得软件容易测试和调试，因此有助于提高软件的可靠性。

7.2.2 抽象化

抽象是一种思维方法。通过这种方法认识事物的时候，人们将忽略事物的细节，通过事物的本质特性来认识事物。具体地说，就是在现实世界中，一定的事物、状态或过程之间总存在着某些相似的共性，把这些相似的方面集中概括起来，暂时忽略它们之间的差异，这就是抽象。在计算机科学中，抽象化(Abstraction)是将数据与程序，以它的语义来呈现出它的外观，但是隐藏起它的实现细节。抽象化用于减少程序的复杂度，使得程序员可以专注于

处理少数重要的部分。一个计算机系统可以被分成几个抽象层(Abstraction Layer),使得程序员可以将它们分开处理。

抽象就是把一个问题或模型,以不同规则或方法得出的不同的解(求解方法和解本身即抽象层)。这些不同的解可以组合并还原成问题或模型的本身。对软件进行模块设计的时候,可以将软件分解为不同的抽象层次,在最高抽象层次上,可以使用问题所处环境的语言描述问题的解法;在较低抽象层次上,可采用过程化的方法,把面向问题的术语和面向实现的术语结合起来描述问题的解法;最后,在最低的抽象层次上,可用直接可以实现的方式描述问题的解法。

1. 过程的抽象

在软件工程过程中,从系统定义到实现,每进一步都可以看做是对软件解决方案的抽象化过程的一次细化。在软件计划阶段,软件被当作整个计算机系统中的一个元素来看待。在软件需求分析阶段,用"问题所处环境的、为大家所熟悉的术语"来描述软件的解决方法。而从概要设计到详细设计的过程中,抽象化的层次逐渐降低,当产生源程序时将到达最低的抽象层次。

2. 数据抽象

数据抽象与过程抽象一样,允许设计人员在不同层次上描述数据对象的细节。例如,可以定义一个 write 数据对象,并将它规定为一个抽象数据类型,用它的构成元素来定义它的内部细节。此时,数据抽象 write 本身是由另外一些数据抽象构成的。而且在定义 write 的抽象数据类型之后,就可以引用它来定义其他数据对象,而不必涉及 write 的内部细节。

3. 控制抽象

控制抽象也可以包含一个程序控制机制而不需要规定其内部细节。控制抽象的例子就是在操作系统中用以协调某些活动的同步信号。

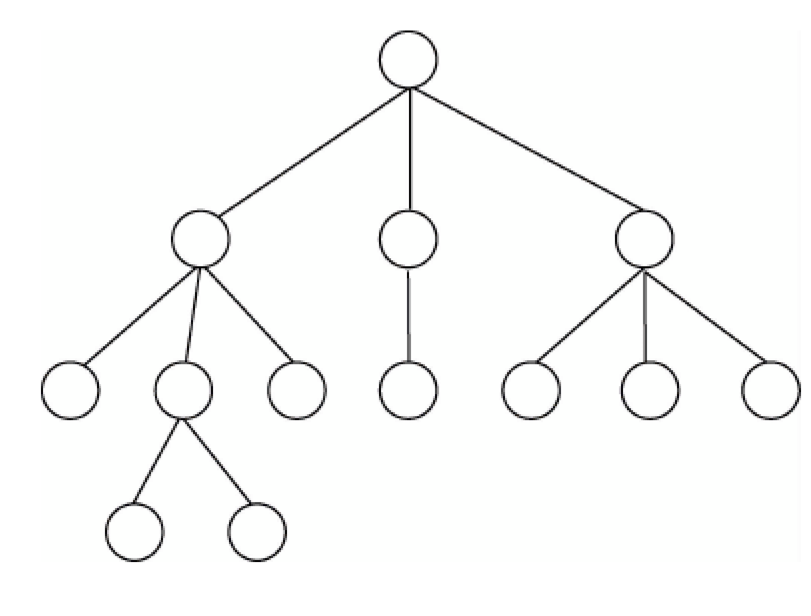
7.2.3 逐步求精

将现实问题经过几次抽象处理,最后到求解域中,只是一些简单的算法描述和算法实现问题。即将系统功能按层次进行分解,每一层不断将功能细化,到最后一层都是功能单一、简单易实现的模块。求解过程可以被划分为若干个阶段,在不同阶段采用不同的工具来描述问题。在每个阶段有不同的规则和标准,产生出不同阶段的文档资料。

逐步求精是由 Niklaus Wirth 最初提出的一种自顶向下的设计策略,是人类解决复杂问题时常采用的一种技术。Wirth 是这样阐述逐步求精过程的:"我们对付复杂问题的最重要的办法是抽象,因此,对一个复杂的问题不应该立刻用计算机指令、数字和逻辑符号来表示,而应该用较自然的抽象语句来表示,从而得出抽象程序。抽象程序对抽象的数据进行某些特定的运算,并用某些合适的记号来表示。对抽象程序做进一步分解,并进入下一个抽象层次,这样的精细化过程一直进行下去,直到程序能被计算机接受为止。这时的程序可能是用某种高级语言或机器指令编写的。"

在人类认识过程中,一般情况下"一个人在任何时候都只能把注意力集中在七个知识块

上”，这就是 Miller 法则。Miller 法则是人类智力的局限，是人类不可能战胜的自然本性。在软件设计中，用户的需求往往不止七个方面，软件的模块数也是远远大于七的，此时逐步求精就变得非常重要。求精就是细化过程，可以将众多的知识块以自顶向下的方式排列展开。软件在设计高抽象级别的功能陈述中，仅仅是概念性地描述了功能，并没有涉及功能内部的工作情况。求精要求设计者逐步细化原始的描述，而随着每个后续求精步骤的完成，会出现越来越多的细节。图 7-2 显示了模块逐步求精的细化过程。


图 7-2 逐步求精

抽象与求精是一对互补的概念，抽象使设计者能够刻画过程和数据，同时却会忽略低层细节。求精则帮助设计者在设计过程中逐步揭示出低层细节。这两个概念都有助于设计者在设计演化过程中构造出完整的设计模型。

逐步求精是人类解决复杂问题时采用的基本方法，也是许多软件工程技术的基础。

7.2.4 信息隐藏和局部化

信息隐藏(Information Hiding)是 D. L. Parnas 于 1972 年提出的把系统分解为模块时应遵循的指导思想。应用模块化原理时，自然会产生一个问题：“为了得到最好的一组模块，应该怎样分解软件?”信息隐藏原理指出：在设计和确定一个模块时，应该让该模块内包含的信息对于不需要这些信息的模块来说，是不能访问的。当程序要调用某个模块时，只需要知道该模块的功能和接口，不需要了解它的内部结构。这就好比我们使用空调，只需要知道如何使用它，而不需要理解空调里面那些复杂的制冷、制热原理和电路图。

局部化的概念和信息隐藏概念是密切相关的，所谓局部化是指把一些关系密切的软件元素物理地放得彼此靠近。在模块中使用局部数据元素是局部化的一个例子，显然，局部化有助于实现信息隐藏。

信息隐藏意味着有效的模块化可以通过定义一组独立的模块来实现，这些独立模块彼此间交换的仅仅是那些为了完成系统功能而必须交换的信息。抽象有利于定义组成软件的过程实体，而隐藏则定义并加强了对模块内部过程细节或模块使用的任何局部数据结构的访问约束。

7.2.5 模块独立性

模块独立的概念是模块化、抽象、信息隐藏和局部化概念的直接结果。所谓模块的独立性，是指软件系统中每个模块只涉及软件要求的具体的子功能，而和其他模块之间没有过多的相互作用。换句话说，若一个模块只具有单一的功能且与其他模块没有太多联系，那么，我们就认为该模块具有独立性。

具有独立性的模块由于接口简单，在软件开发过程中比较容易被开发，在测试时也容易被测试和维护。

一般使用两个定性标准来衡量模块的独立程度：耦合和内聚。耦合用于衡量不同模

块彼此间互相依赖的紧密程度；内聚用于衡量一个模块内部各个元素彼此结合的紧密程度。

1. **耦合**

耦合是一个软件结构内不同模块之间互连程度的度量。简单地说，软件工程中对象之间的耦合度就是对象之间的依赖性。在软件设计中应该追求实现尽可能松散耦合的系统，这样开发、测试任何一个模块，不需要对系统的其他模块有太多的了解，如果一个模块发生错误，影响其他模块的可能性就很小。所以，模块耦合越高，维护成本越高。因此软件的设计应使模块之间的耦合最小。

耦合性是程序结构中各个模块之间相互关联的度量。它取决于各个模块之间的接口的复杂程度、调用模块的方式以及哪些信息通过接口。如果有两个模块，每一个模块都能独立的工作，而不需要另一个模块，那么它们之间是完全独立的，它们的耦合程度最低。但是一个系统中不可能所有的模块之间都没有任何联系。

耦合可以分为以下几种，它们之间的耦合度由高到低排列如下。

(1) 内容耦合：当一个模块直接修改或操作另一个模块的数据时，或一个模块不通过正常入口而转入另一个模块时，这样的耦合被称为内容耦合。内容耦合是最高程度的耦合，应该避免使用它。

(2) 公共耦合：两个或两个以上的模块共同引用公共数据环境的一个全局数据项，这种耦合被称为公共耦合。在具有大量公共耦合的结构中，确定究竟是哪个模块给全局变量赋了一个特定的值是十分困难的。公共数据环境包括全局变量、共享的通信区、内存的公共覆盖区、任何存储介质上的文件、物理设备等。

(3) 控制耦合：一个模块通过接口向另一个模块传递一个控制信号，接受信号的模块根据信号值而进行适当的动作，这种耦合被称为控制耦合。控制耦合是中等程度的耦合，它增加了系统的复杂程度。控制耦合往往是多余的，在把模块适当分解之后通常可以用数据耦合代替它。

(4) 特征耦合：当模块之间传递的是某些数据结构，但是目标模块只是使用了数据结构中的部分内容时，这种耦合方式称为特征耦合。例如模块 A 给模块 B 传递某个书对象时，模块 B 只是使用了该对象的一个书号属性，那么模块 A 与模块 B 就是特征耦合，此时应该把模块 A 给模块 B 传递的参数改为某书的书号，将特征耦合变为数据耦合。

(5) 数据耦合：模块之间通过参数来传递数据，那么被称为数据耦合。数据耦合是最低程度的一种耦合形式，系统中一般都存在这种类型的耦合，因为为了完成一些功能，往往需要将某些模块的输出数据作为另一些模块的输入数据。

(6) 非直接耦合：两个模块之间没有直接关系，它们之间的联系完全是通过主模块的控制和调用来实现的。

耦合是影响软件复杂程度和设计质量的一个重要因素，在设计中应采用以下原则：如果模块间必须存在耦合，就尽量使用数据耦合，少用控制耦合，限制公共耦合的范围，完全不用内容耦合。

2. 内聚

内聚标志一个模块内各个元素彼此结合的紧密程度，它是信息隐蔽和局部化概念的自然扩展。内聚是从功能角度来度量模块内的联系，一个好的内聚模块应当恰好做一件事。内聚的概念是由 Constantine、Yourdon、Stevens 等人提出的。按他们的观点，把内聚按紧密程度(强度)从低到高排列的次序为偶然内聚、逻辑内聚、时间内聚、过程内聚、通信内聚、顺序内聚、功能内聚。

(1) 偶然内聚：如果一个模块的各成分之间毫无关系，则称其为偶然内聚。也就是说模块完成一组任务，这些任务之间的关系松散，实际上没有什么联系。很多软件设计新手都喜欢把多个本来功能不相干的模块组合在一起形成一个模块，仅仅是为了设计程序上的方便，但是这种偶然内聚会导致软件结构不清晰，难以理解和调试，也为后续模块重用带来了麻烦。

(2) 逻辑内聚：几个逻辑上相关的功能被放在同一模块中，则称其为逻辑内聚。如调用模块在每次调用时传递一个“读”或“写”参数给被调用模块，被调用模块根据该参数选择是“读”一个记录还是“写”一个记录，那么这个被调用模块就属于逻辑内聚。逻辑内聚也会导致模块结构不清晰，难以理解、调试及重用，应把“读”功能和“写”功能分解开，分别形成两个独立的模块。

(3) 时间内聚：如果一个模块完成的功能必须在同一时间内执行(如系统初始化)，但这些功能只是因为时间因素关联在一起，则称其为时间内聚。

(4) 过程内聚：如果一个模块内部的处理是相关的，而且这些处理必须以特定的次序执行，则称其为过程内聚。使用程序流程图作为工具设计软件时，常常通过研究流程图确定模块的划分，这样得到的往往是过程内聚的模块。

(5) 通信内聚：如果一个模块的所有元素都使用同一个输入数据和产生同一个输出数据，则称其为通信内聚。例如，某模块要求根据“书号”查询所有书的价格，再根据“书号”更改新书的最新数量，这两个处理动作都使用了相同的输入数据“书号”，那么该模块是通信内聚。可以把相同输入或相同输出的这些功能分解为多个模块，以提高内聚程度。

(6) 顺序内聚：如果一个模块的处理元素和同一个功能密切相关，而且这些处理必须按某种顺序执行，则称其为顺序内聚。顺序内聚是一部分的输出是另一部分的输入，显然，如果上一部分没有完成，下一部分是不可能执行的。

(7) 功能内聚：模块的所有成分对于完成单一的功能都是必需的，则称其为功能内聚。软件结构中应多使用功能内聚模块。

耦合是软件结构中各模块之间相互连接的一种度量，耦合强弱取决于模块间接口的复杂程度、进入或访问一个模块的点以及通过接口的数据。程序讲究低耦合、高内聚。就是同一个模块内的各个元素之间要高度紧密，但是各个模块之间的相互依存度却不要过于紧密。内聚和耦合是密切相关的，与其他模块存在高耦合的模块意味着低内聚，而高内聚的模块意味着该模块与其他模块之间是低耦合的。

实践证明，内聚比耦合更为重要，我们应该把更多的注意力集中到提高模块的内聚度上来。

7.2.6　模块层次化

层次表明了程序模块的组织情况，位于最上层根部的是顶层模块，它是程序的主要模块，与其联系的有若干个下属模块，各下属模块还可以进一步引出更下一层的下属模块。

- 程序结构的深度：程序结构的层次数称为结构的深度。结构的深度在一定意义上反映了程序结构的规模和复杂程度。
- 程序结构的宽度：层次结构中同一层模块的最大模块个数称为结构的宽度。
- 模块的扇入和扇出：扇出表示一个模块直接调用的其他模块数目；扇入则被定义为调用一个给定模块的模块个数。多扇出意味着需要控制和协调许多下属模块。而多扇入的模块通常是公用模块。

图 7-3 为教务管理系统的模块层次。

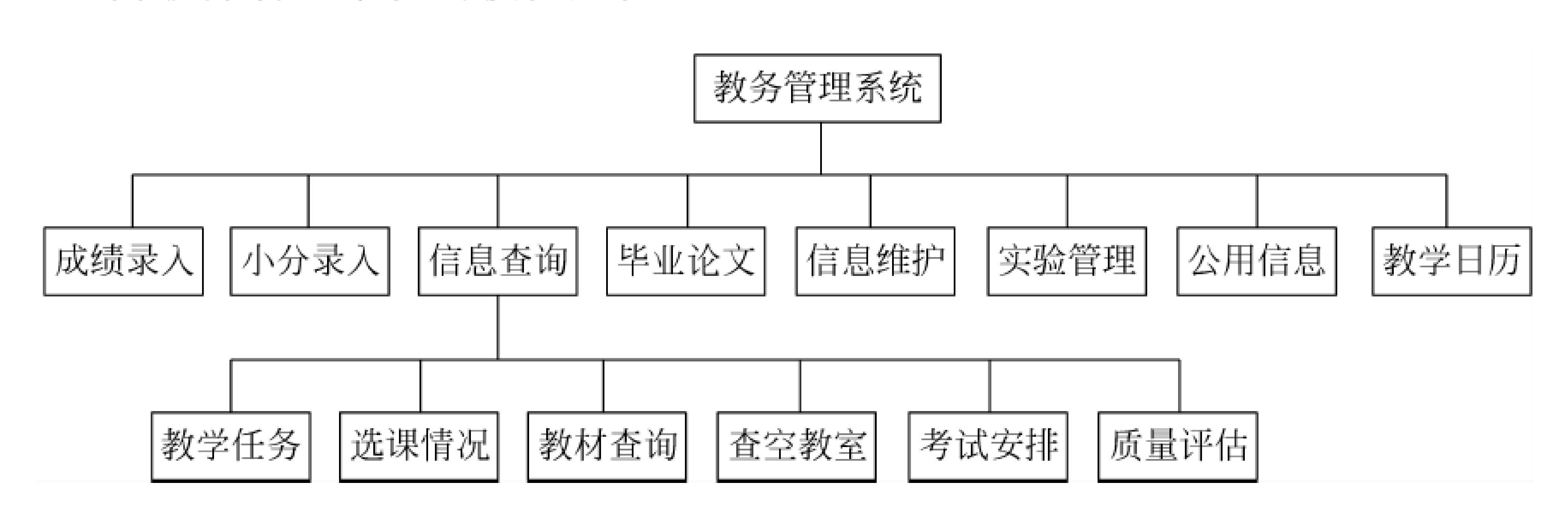


图 7-3　教务管理系统模块层次

一般采用树状结构来清晰表示层次结构，允许上层模块调用下层模块，但不允许下层模块调用上层模块，应避免上层模块越级调用下层模块。

7.2.7　启发式规则

人们在开发计算机软件的实践中总结了丰富的经验，汇总得出模块设计的启发式规则。这些规则能够帮助我们找到改进软件设计、提高软件质量的途径。

调整软件结构以提高模块独立性。通过模块的分解或合并，可提高模块间的内聚度，降低模块间的耦合度。

软件结构的深度、宽度、扇入数和扇出数应该适当。深度和宽度能够粗略地反应一个系统的大小和复杂程度，如果当深度或宽度过大时，应该考虑合并部分模块。

模块的影响范围应该在控制范围之内。模块的影响范围是指所有受该模块的运行所影响的模块的集合。模块的控制范围是指所有直接或间接被该模块调用的模块集合。一个好的模块化设计，应当是某模块的运行仅仅影响那些被该模块直接或间接调用的模块。

应降低模块接口的复杂程度。软件错误常常发生在模块接口处，应仔细设计模块接口，使得信息传递简单并且和模块的功能一致。

模块功能应该是可以预期的。模块功能可以预期是指某模块可以作为黑盒子来对待，开发者在使用该模块时，可以不考虑内部处理的细节。带有内部“存储器”的模块的功能可能是不可预测的，因为它的输出可能取决于内部存储器的状态，由于内部存储器对于上级模

块是不可见的，因此这样的模块是不可预期的，在使用时应该加以注意。

7.3 面向过程设计

过程设计是为了获得高质量软件所需要完成的一系列任务的框架，它规定了完成各项任务的工作步骤。

描述程序处理过程的工具称为过程设计工具，它们可以分为图形、表格和语言三类。不论是哪类工具，对它们的基本要求都是能提供对设计的无歧义地描述，也就是应该能指明控制流程、处理功能、数据组织以及其他方面的实现细节，从而在编码阶段能把对设计的描述直接翻译成程序代码。此外，这类工具应该尽可能形象直观、易学、易懂。

1. 程序流程图

程序流程图又称程序框图，是使用最广泛的、描述过程设计的方法。它的优点是对控制流的描绘很直观，便于初学者掌握。程序流程图中的符号有确切的规定，除了使用规定的符号外，不允许出现其他任何符号。图 7-4 给出由国际标准化组织提出的，已获得我国国家技术监督局批准的一些程序流程图标准符号。

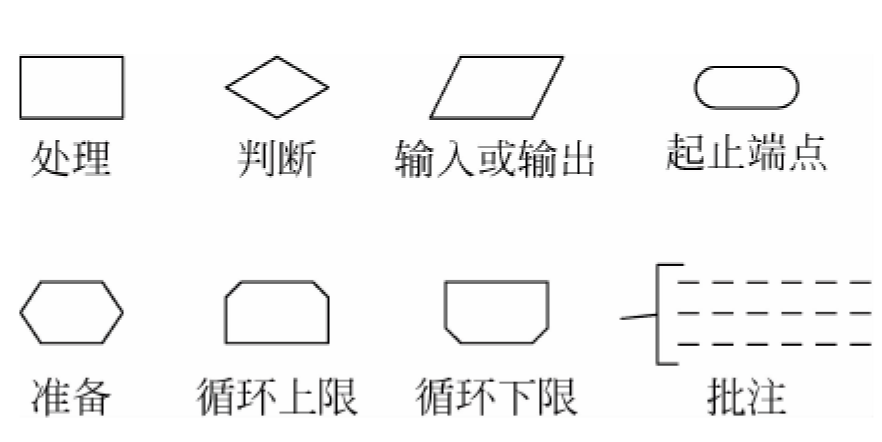


图 7-4　程序流程图标准符号

使用程序流程图描述结构化程序，就必须限制流程图只能使用图 7-5 所给出的五种基本控制结构。

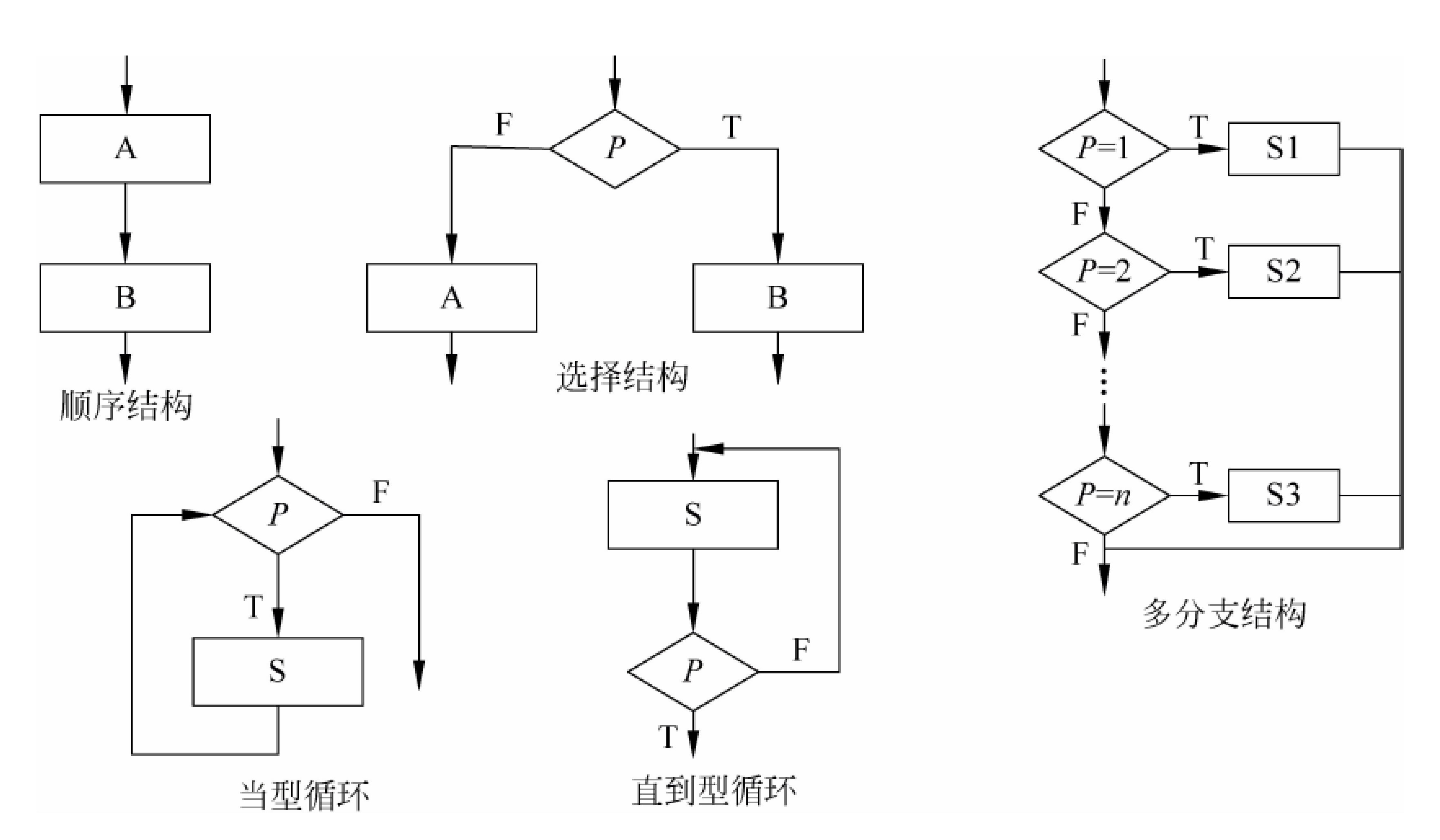


图 7-5　流程图的基本控制结构

程序流程图的主要缺点有以下三个：

(1) 程序流程图本质上不是逐步求精的好工具，它诱使程序员过早地考虑程序的控制流程，而不去考虑程序的全局结构。

（2）程序流程图用箭头代表控制流，容易使程序员不受任何控制，随意转移控制。
（3）程序流程图不易表示数据结构。

2. N-S 图

Nassi 和 Shneiderman 发明了 N-S(Box-Diagram，又称盒图)图，N-S 图没有箭头，不能够随意转移控制。使用 N-S 图可以使程序员逐步养成用结构化的方式思考问题、解决问题的习惯。图 7-6 给出了 N-S 图的使用实例。

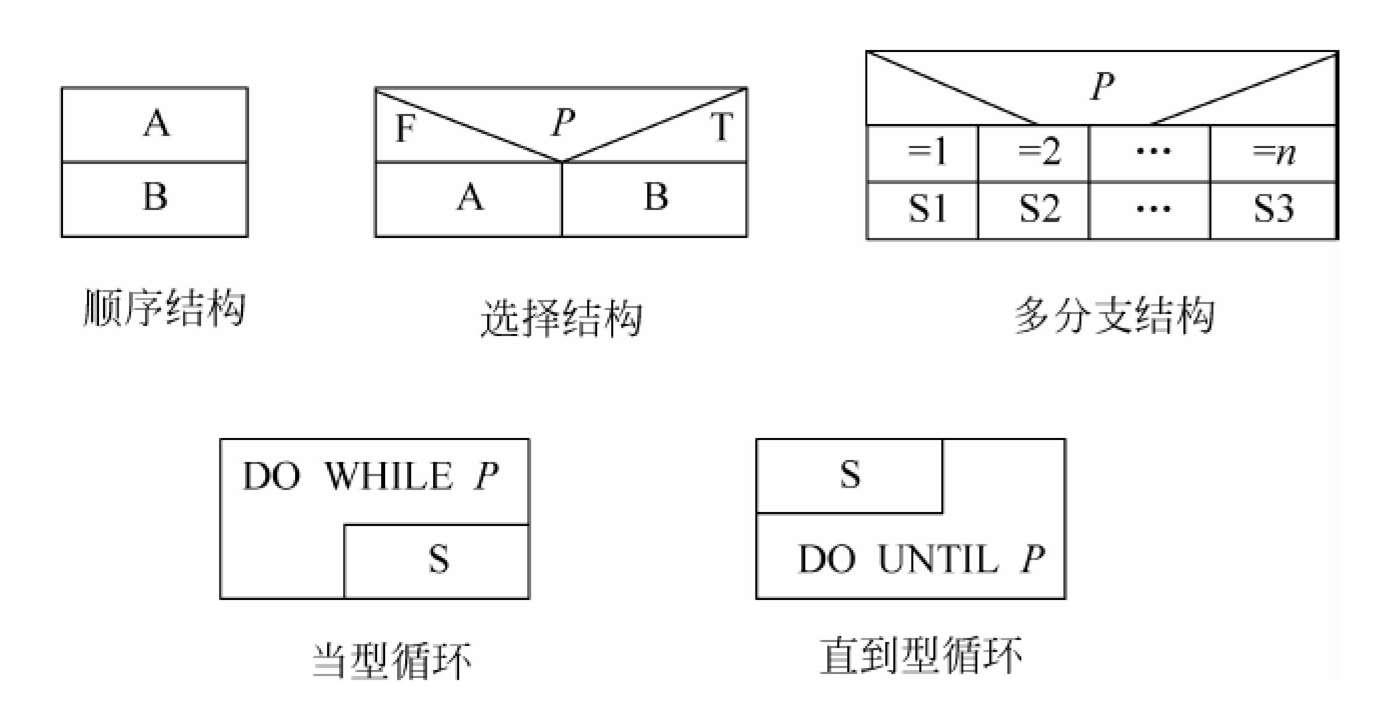


图 7-6　N-S 图

N-S 图有以下五个特点。
（1）功能域明显。
（2）不能实现任意转移。
（3）很容易体现局部和全局的作用域。
（4）很容易表示嵌套结构和模块的层次结构。
（5）清晰表示模块间的调用关系。

3. 问题分析图

PAD 是问题分析图 Problem Analysis Diagram 的英文缩写，是日本日立公司于 1973 年提出的一种主要用于描述详细设计的图形表示工具。它用二维树形结构的图来表示程序的控制流，将这种图翻译成程序代码是比较容易的。图 7-7 给出了 PAD 基本的控制结构。

PAD 图的优点有以下六个：
（1）设计的程序一定是结构化程序，算法描述最好使用 PAD 图。
（2）描述的流程图比较清晰。图中最左边的竖线是程序的主线，随着程序层次的增加，PAD 图逐渐向右延伸，每增加一个层次，图形向右扩展一条竖线。PAD 图中竖线的总条数就是程序的层次数。
（3）表示程序的逻辑结构易懂、便于记忆。PAD 图是二维树形结构的图形，程序从图中最左边竖线上端的结点开始执行，自上而下、从左至右顺序执行，遍历所有结点。
（4）可以很容易地把 PAD 图转化为高级语言源程序。这种转换可用软件工具自动完成，从而可省去人工编码的工作，有利于提高软件可靠性和软件生产率。

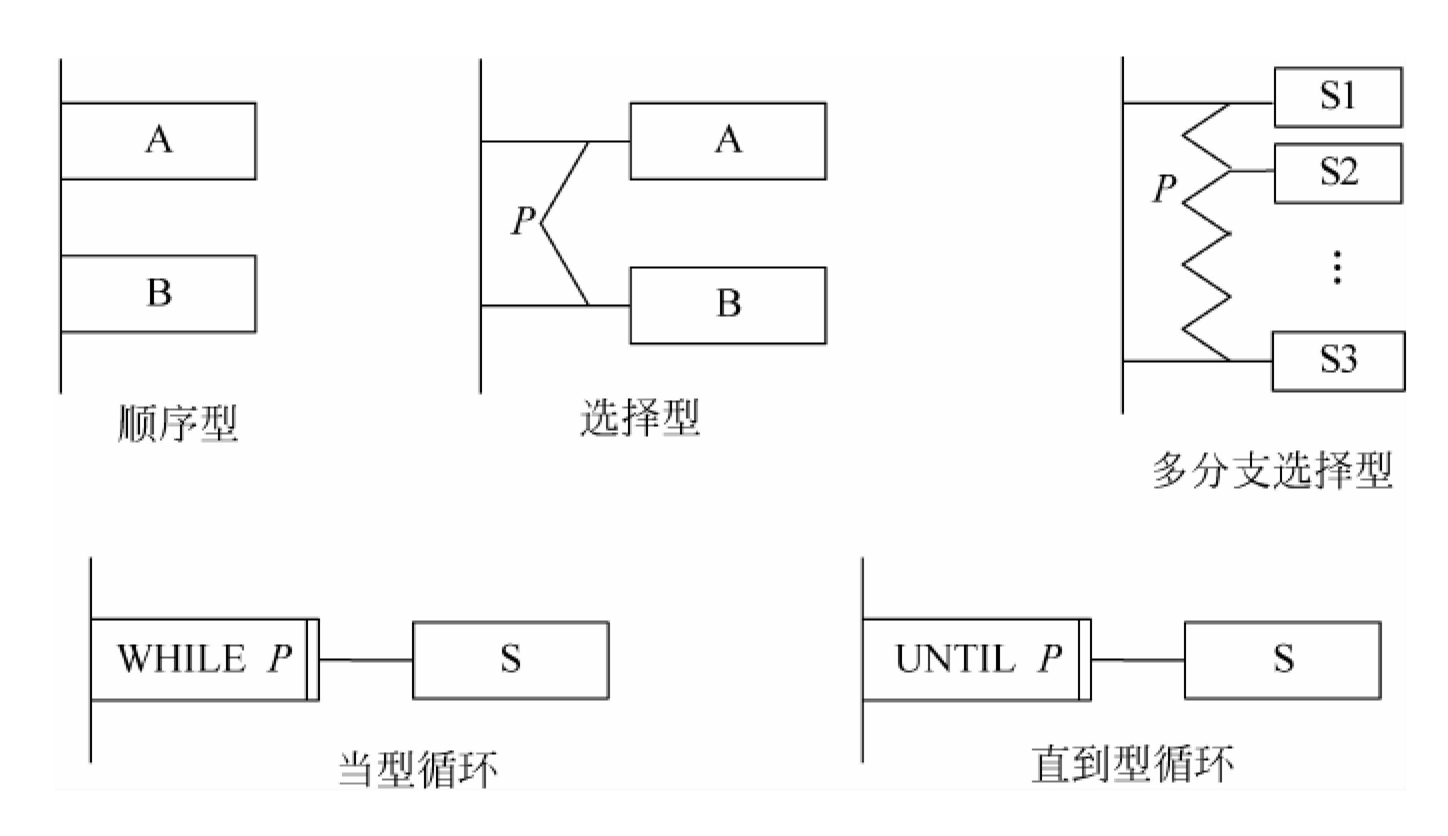


图 7-7　PAD 基本控制结构

（5）可用于描述数据结构。

（6）支持自顶向下，逐步求精。在开始时，设计者可以定义一个抽象的程序，随着设计工作的深入而使用 def 符号逐步增加细节，直至完成详细设计。

4. 判定表

当算法中包含多重嵌套的条件选择时，用程序流程图、N-S 图、PAD 图都不容易清晰地描述，用判定表却能够清晰地表示复杂的条件组合与应做的动作之间的对应关系。

一张判定表由四部分组成，左上部列出所有条件，左下部是所有可能做的动作，右上部是表示各种条件组合的一个矩阵，右下部是和每种条件组合相对应的动作。判定表右半部的每一列实质上是一条规则，规定了与特定的条件组合相对应的动作。表 7-1 显示了判定表的一般结构。

表 7-1　判定表的一般结构

	1	2	3	4
C1	T	T	F	F
C2	T	F	T	F
A1	√		√	
A2		√		
A3				√

判定表的优点是能够简洁且无二义性地描述所有处理规则。但判定表表示的是静态逻辑，是在某种条件组合情况下可能的结果，它不表达加工的顺序，也不能显示循环结构。因此，判定表不适用于作为一种通用的设计工具。

5. 判定树

判定表虽然能清晰地表示复杂的条件组合与应做的动作之间的对应关系，但其含义并不是一眼就能被看出来的。判定树是判定表的变种，也能清晰地表示复杂的条件组合与应做的动作之间的对应关系。判定树的优点在于，它的形式简单到不需要进行任何说明，一眼就可以看出其含义，因此易于掌握和使用。

多年来，判定树一直受到人们的重视，是一种比较常用的系统分析和设计的工具。从图 7-8 可以看出，某幼儿园的游戏分配方案是：年龄在 3 岁以下的人，男孩玩跷跷板，女孩玩老鹰捉小鸡；年龄在 3 岁～5 岁的人，男孩玩跳床，女孩玩手工制作；年龄大于 5 岁的人参加拔河比赛。

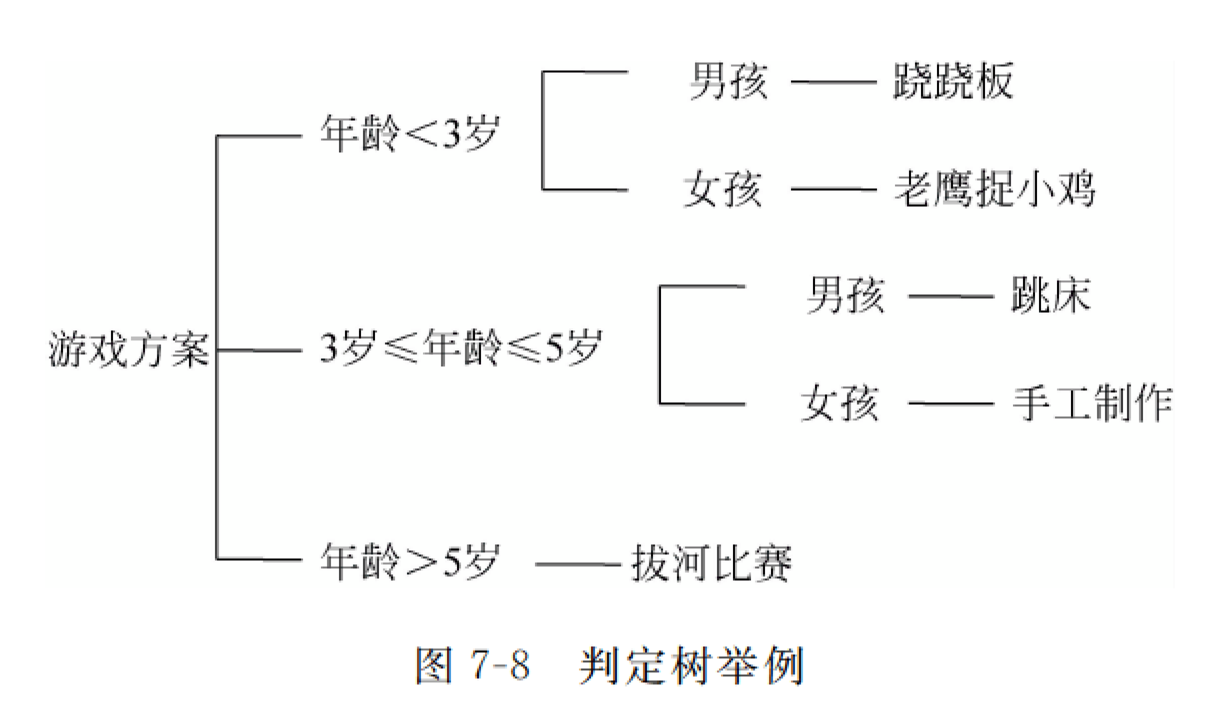


图 7-8　判定树举例

虽然判定树比判定表更直观，但简洁性却不如判定表，数据元素的同一个值往往要重复写多遍，而且越接近叶端点重复次数越多。

6. 过程设计语言 PDL

PDL(Program Design Language)也称为伪代码，是一种用于描述模块算法设计和数据表示的语言。一般情况下，伪码的语法规则分为外语法和内语法。外语法应当符合一般程序设计语言常用语句的语法规则；而内语法可以用英语中一些简单的句子、短语和通用的数学符号来描述程序应执行的功能。外语法规定严格，用于定义控制结构和数据结构；内语法则灵活自由，用于描述实际的处理过程。

由于 PDL 可以加入自然叙述文字，因此它不能被直接翻译成程序语言。一个程序如果具有以下特点，可以认为它是 PDL：

(1) 有固定格式的关键字语法，提供了结构化的控制结构、数据说明和模块化特征。这些固定的关键词能够很好地对整个模块结构进行划分，使得整个模块结构清晰、可读性好。

(2) 内语法允许使用自然语言，这些自然语言可以是程序、公式、句段等文字。

(3) 数据声明可以说明各种数据结构，既包括简单的数据结构，如数组；又包括复杂的数据结构，如链表。

(4) 模块定义和调用的技术，应该提供各种接口的描述模式。

下面是一个 PDL 的例子，其功能是查找拼写错误的单词。

```
PROCEDURE spell_check
  BEGIN
      Split document into single words
      Look up words in dictionary
      Display words which are not in dictionary
      Create a new dictionary
  END spell_check
```

用 PDL 编写的伪码程序还可以做到逐步求精，从比较概括和抽象的 PDL 程序着手，逐

步写出更详细、更精确的描述。如可将前面给出的代码看做是一个框架，为进一步了解查找拼写错误单词的方法，可以对上面四个步骤的每一步进行细化。

```
PROCEDURE spell_check
  BEGIN
      /* Split document into single words */
      Loop
          Get next word
          Add word to word list in sort order
          EXIT WHEN all words processed
      END LOOP
      /* Look up words in dictionary */
Loop
    Get word from word list
    IF word not in dictionary THEN
              /* Display words which are not in dictionary */
              display word,prompt on user terminal
              IF user response say word OK THEN
                  Add word to good word list
              ELSE
                  Add word to bad word list
          ENDIF
          EXIT WHEN all words processed
      END LOOP
      /* Create a new dictionary */
      dictionary := merge dictionary and good word list
  END spell_check
```

PDL 作为一种设计工具有以下三个优点：

(1) 可以作为注释直接插在源程序中间。这样做能使维护人员在修改程序代码的同时也能相应修改 PDL 注释，有助于保持文档和程序的一致性，提高了文档的质量。

(2) 可以使用普通的文本编辑程序或文字处理系统，很方便地完成 PDL 的书写和编辑工作。

(3) 已经有自动处理 PDL 的程序，而且可以自动由 PDL 生成程序代码。

PDL 的缺点是不如图形工具形象直观。在描述复杂的条件组合与动作间的对应关系时，不如判定表清晰简单。

7.4 面向对象设计

面向对象方法的出现以 20 世纪 60 年代后期挪威奥斯陆大学和挪威计算中心共同研制的 SIMULA 语言的出现为标志，面向对象技术是近三十年来获得广泛应用的一种具有广阔发展前景的技术。面向对象方法的出发点和基本原则，是尽可能模拟人类习惯的思维方式，使开发软件的方法与过程尽可能接近人类认识世界、解决问题的方法与过程。

面向对象方法是一种自底向上和自顶向下相结合的方法，它以对象建模为基础，不仅考虑了输入、输出数据结构，实际上也包含了所有对象的数据结构。OO 技术在需求分析、可

维护性和可靠性这三个软件开发的关键环节和质量指标上有了实质性的突破。

面向对象方法的特点有以下五个。

(1) 自底向上地归纳。

(2) 自顶向下地分解。

(3) 基础是对象模型。

(4) 需求分析彻底。

(5) 可维护性大大改善。

7.4.1 面向对象方法概述

在采用传统软件工程的方法开发软件时,主要存在以下三个问题:

(1) 软件重用性差。重用性是指同一事物不经修改或稍加修改就可以多次重复使用的性质。

(2) 软件可维护性差。实践证明,用传统方法开发出来的软件,其维护费用和成本仍然很高,其原因是可修改性差、维护困难。

(3) 开发出的软件不能满足用户需求。用传统的结构化方法开发的软件,其稳定性、可修改性和可重用性都比较差,这是因为结构化方法的本质是功能分解,从代表目标系统整体功能开始,自顶向下不断把复杂的处理分解成子处理,这样一层层地分解下去,直到仅剩容易实现的子处理功能为止,然后用相应的工具来描述各个最底层的子处理。因此,结构化方法是围绕实现功能的"过程"来构造系统的。而用户的需求变化大部分是针对功能的,用这种方法设计出来的系统结构常常是不稳定的,用户需求的变化往往造成系统结构的较大变化,从而需要花费很大代价才能实现这种变化。

面向对象方法学的优点有以下五个:

(1) 它与人类习惯的思维方式一致。面向对象方法使用现实世界的概念抽象的思考问题从而自然地解决问题,它强调模拟现实世界中的概念而不强调算法。面向对象方法是以对象为核心,对象是对现实世界实体的抽象,它是由描述内部状态表示静态属性的数据,以及可以对这些数据施加的操作(表示对象的动态行为)封装在一起所构成的统一体。对象之间通过传递消息相互联系来模拟现实世界中不同事物彼此之间的联系。

(2) 稳定性好。面向对象方法基于构造问题领域的对象模型,以对象为中心构造软件系统。它的基本做法是用对象模拟问题领域中的实体,以对象间的联系刻画实体间的联系。因为面向对象的软件系统结构是根据问题领域的模型建立起来的,所以,当对系统的功能需求变化时并不会引发软件结构的整体变化,往往仅需要做一些局部修改。

(3) 可重用性好。面向对象的软件技术在利用可重用的软件成分构造新的软件系统时,有很大的灵活性。有两种方法可以重复使用一个对象类:一种是创建该类的实例,从而直接使用它;另一种是从它派生出一个满足当前需要的新类。继承性使得子类不仅可以继承父类的数据结构和程序代码,而且可以在父类的基础上对其进行方便地修改和扩充,这种修改并不影响对原有类的使用。

(4) 可实现对大型软件产品的开发。用面向对象方法学开发软件时,构成软件系统的每个对象就像一个微程序,有自己的数据、操作、功能和用途。因此,可以把一个大型软件产品分解成一系列本质上相互独立的小产品来处理,不仅降低了开发的技术难度,也使得开发

工作的管理变得容易许多。经验表明，使用面向对象方法学开发大型软件，软件成本明显降低了，软件的整体质量也提高了。

(5) 可维护性好。面向对象的软件比较容易理解，稳定性好、易于修改、易于测试和调试。

7.4.2 面向对象的概念

面向对象的概念有：类、对象、抽象、继承、封装、多态等。要掌握面向对象的技术和方法，必须明确“对象”和“面向对象”的概念，Coad 和 Yourdon 给出了一个简洁的定义：

面向对象(Object-Oriented)＝对象(Object)＋类(Class)＋继承(Inheritance)＋通信

如果一个软件系统是使用这样四个概念设计和实现的，则将该软件系统称为面向对象的。

1. 对象

对象是现实世界中一个实际存在的事物。它可以是有形的，比如一个学生、一辆汽车、一本图书；也可以是无形的，比如一项计划、一个贷款。对象构成世界的一个独立单位，它具有自己的静态特征和动态特征。静态特征是可以用某种数据来描述的特征，动态特征即对象所表现的行为或对象所具有的功能。

对象是由数据和允许的操作组成的封装体，与客观实体有直接对应关系，一个对象类定义了具有相似性质的一组对象。对象是要研究的任何事物，不仅能表示有形的实体，也能表示无形的、抽象的规则、计划或事件。对象由数据和作用于数据的操作构成一个独立整体。从程序设计者来看，对象是一个程序模块，从用户角度来看，对象为他们提供所希望的行为。使用对象时只需要知道它向外界提供的接口形式，不需要知道它的内部实现算法。对象的使用非常简单、方便，具有很高的安全性和可靠性。对象内部的数据只能通过对象的公有方法来访问或处理，这就保证了对这些数据的访问和处理在任何时候都是使用统一的方法进行的。

对象是封装了数据结构及可以施加在这些数据结构上的操作的封装体，这个封装体有可以唯一标识它的名字，而且向外界提供一组服务。对象具有以下几个特点。

(1) 以数据为中心。操作围绕对其数据所需要做的处理来设置，不设置与这些数据无关的操作，而且操作的结果往往与当时所处的状态有关。

(2) 对象是主动的。对象与传统的数据有本质的不同，它不是被动地等待处理，而是处理的主体。为了完成某项操作，不能从外部直接访问它的私有数据，必须通过它的公有接口向对象发消息，请求执行某项操作，访问它的私有数据。

(3) 实现了数据封装。对象好比是一个黑盒子，它的私有数据放在盒子内部，外面是看不见盒子内部的，对盒子内私有数据的访问只能通过公有的操作进行。为了访问对象内部的私有数据，只需要知道数据的取值范围和可以对该数据施加的操作，不需要知道数据的具体结构以及实现操作的算法，这就是抽象数据类型的概念。

(4) 本质上具有并行性。不同的对象各自独立处理自身的数据，彼此通过发消息传递信息完成通信，因此，具有并行工作的属性。

(5) 模块独立性好。模块的独立性要求模块内聚性强、耦合性弱，对象内部彼此结合得

很紧密，内聚性很强，由于完成对象功能所需要的数据和操作基本上都被封装在对象内部，它与外界的联系自然比较少，所以，对象之间的耦合性弱。

2. 类

在面向对象的软件技术中，类就是一组具有相同属性和相同操作的对象的集合。一个类中的每个对象都是这个类的一个实例。类是创建对象的模板，从同一个类实例化的每个对象都具有相同的结构和行为。类和对象的关系非常密切，可以说类是支持继承的抽象数据类型，而对象则是类的实例。

例如，张三、李四、王五……每个人有不同的姓名、学号、性别、籍贯，但他们的基本特征是相似的，都是学生，于是可以把他们统称为“学生类”。这里的“张三”就是一个对象，“学生类”就是抽象出来的类，类在现实世界中并不能真实存在，没有人是抽象出来的“学生类”，类是建立对象时使用的“样板”，用这个样板建立的一个个具体的对象就是张三、李四……

3. 消息

消息是传递时对象间通信的手段，一个对象通过向另一个对象发送消息来请求其服务。一个消息通常包括接收对象名、调用的操作名和适当的参数。消息只告诉接收对象需要完成什么操作，但并不指示接收者怎样完成操作，消息完全由接收者解释执行。

4. 封装

封装是面向对象方法的一个重要原则，它有两个含义。第一个含义是把对象的全部属性和全部操作结合在一起，形成一个不可分割的独立单位即对象；第二个含义是尽可能地隐蔽对象的内部细节，对外形成一个边界，只保留有限的接口与外部发生联系。

封装是一种信息隐蔽技术，用户只能看见对象封装界面上的信息，对象的内部实现对用户是隐蔽的，不能从外面直接访问或修改这些数据和代码。封装的目的是使对象的使用者和生产分离，使对象的定义和实现分开。

对象具有封装性的条件有以下三个：

(1) 有一个清晰的边界。所有私有数据和实现操作的代码都被封装在这个边界内，从外部是看不见的，也不能直接访问。

(2) 有确定的接口。这些接口可以接受外部的消息，对象间传递消息是通过接口实现的。

(3) 受保护的内部实现。实现对象功能的细节不能在定义该对象的类的范围外访问。

5. 继承

继承是指能够直接获得已有的性质和特征，而不必重复定义它们，在面向对象的软件技术中，继承是子类自动地共享基类中定义的数据和方法的机制。

继承是类间的基本关系，它是基于层次关系的不同类共享数据和操作的一种机制。父类中定义了其所有子类的公共属性和操作，在子类中除了定义自己特有的属性和操作外，可以继承其父类的属性和操作，还可以对父类中的操作重新定义其实现方法。继承性使得相似的对象可以共享程序代码和数据结构，从而大大降低了程序的冗余性。

继承具有传递性，如果 A 类继承了 B 类，B 类继承了 C 类，那么 A 类也就继承了 C 类。因此，一个类实际上继承了它所在等级中所有上层基类描述。当一个类只允许有一个父类时，这种继承为单继承；当允许一个类有多个父类时，这种继承是多重继承。多重继承的类可以组合多个父类的性质构成所需要的性质，因此功能更强大。但在使用多重继承时要注意避免二义性。

6. 多态性

在面向对象的软件技术中，多态性是指子类对象可以像父类对象那样使用，同样的消息既可以发送给父类对象也可以发送给子类对象。也就是说，在类等级的不同层次中，可以共享一个行为的名字，然而不同层次中的每个类却各自按自己的需要来实现这个行为。当对象接收到发送给它的消息时，根据该对象所属于的类动态选择在该类中定义的实现算法。举一个例子，假如有一个函数负责某人来吃饭，函数要求传递的参数是人这个对象。如果来了个中国人，就是用筷子在吃饭；如果来了个英国人，则拿着刀叉在吃饭。这就体现了同样一个方法，却可以产生不同的形态，这就是多态性。

在我们学过的 C++ 语言中，多态性是通过虚函数实现的。在不同等级的类中名字、参数特征和返回值类型都相同的虚拟成员函数，这些虚函数实现算法各不相同。这使得程序员能在一个类等级中使用相同函数的多个不同版本，在运行时根据接收消息的对象所属的类，决定执行哪个版本。

7. 重载

重载可以分为函数重载和运算符重载。函数重载是指在同一作用域内的若干个参数特征不同的函数可以使用相同的函数名字；运算符重载是指同一个运算符可以施加于不同类型的操作数上面。当然，当参数特征不同或被操作数的类型不同时，实现函数的算法或运算符的语义是不相同的。

面向对象的分析是一种研究问题域的过程，该过程产生对外部可见行为的描述。面向对象的设计是指在分析的描述基础上，加入实际计算机系统实现所需要的细节的过程。从面向对象分析到面向对象设计，是一个逐渐扩充模型的过程，或者说，面向对象设计就是以面向对象观点建立求解域模型的过程。

7.4.3 面向对象的模型

模型就是为了理解事物而对事物做出的一种抽象，是对事物的一种无歧义的书面描述。模型通常由一组规定的符号和组织这些符号的规则组成，利用它们来定义和描述问题域。模型可以帮助人们思考问题，有助于开发复杂的软件系统。

使用面向对象方法开发软件，通常需要建立三种形式的模型，它们分别是：对象模型、动态模型和功能模型。用对象模型可描述系统数据结构，用动态模型可描述系统控制结构，用功能模型可描述系统功能。这三种模型都涉及数据、控制、操作等共同的概念，只不过每种模型描述的侧重点不同，将三个模型综合起来则可全面反应系统的需求。

1. 对象模型

对象模型是三个模型中最关键的模型，它表现了静态的、结构化的系统数据性质，描述了系统的静态结构，它从客观世界实体对象的关系角度来描述，表现了对象的相互关系。

关联是建立类与类之间的关系，链是建立对象与对象之间的关系，关联是链的抽象，链是关联的实例。两个类之间的关联称为二元关联，用一条直线表示。三个类之间的关联称为三元关联，应在三个类之间的连线上画一个菱形符号。图 7-9 为二元关联和三元关联的表示方法。

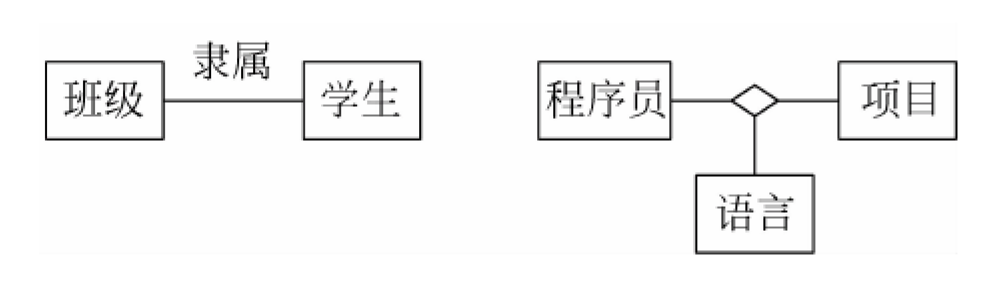


图 7-9 二元关联和三元关联

受限关联由两个类及一个限定词组成，限定词是一种特定的属性，用来有效地减少关联的重数，限定词在关联的终端对象集中说明。限定关联通常用在一对多或多对多的关联关系中，可以把模型中的重数从一对多变成一对一。例如，一个目录下有许多文件，一个文件仅属于一个目录，在一个目录内文件名确定了唯一一个文件，这里的文件名就是限定词，利用文件名这个限定词把一对多关系简化为一对一关系。受限关联的表示方法如图 7-10 所示。

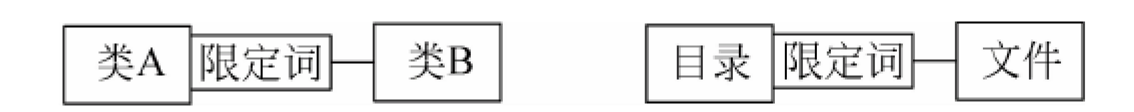


图 7-10 受限关联的表示方法

关联的多重性是指类中有多少个对象与关联的类的一个对象相关。关联重数可用对象图关联连线的末端的特定符号表示，小实心圆表示"多个"，小空心圆表示"0 个"或"1 个"，没有符号表示一对一关联，关联重数的表示方法如图 7-11 所示。

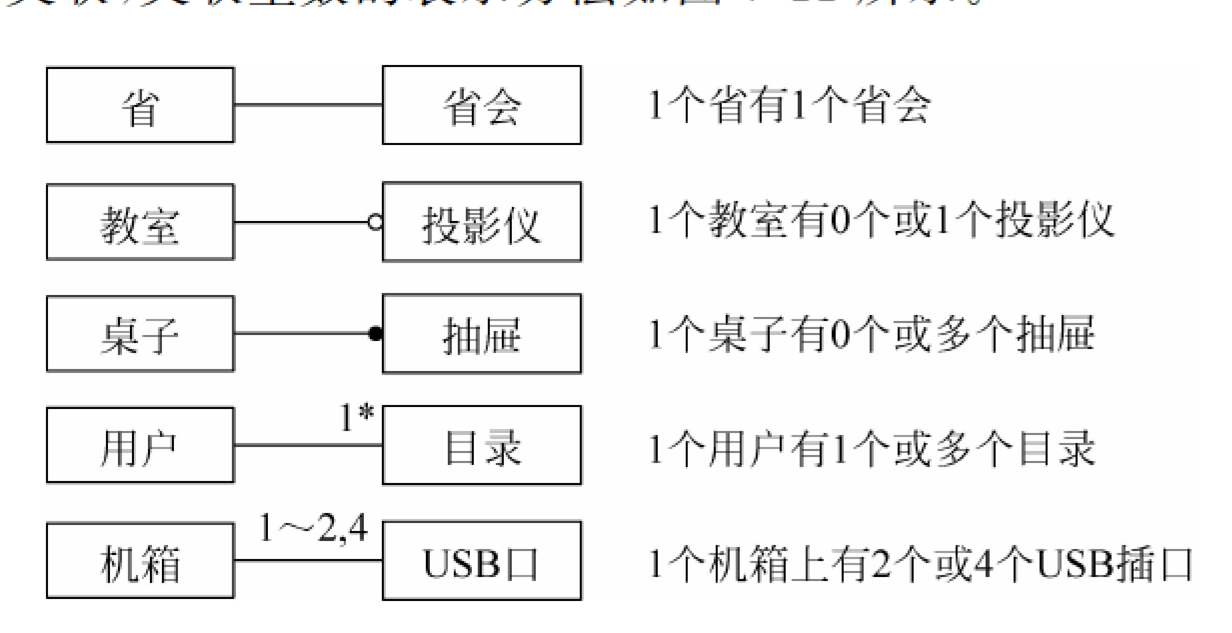


图 7-11 关联重数的表示方式

类的层次结构有聚集和一般化关系。聚集表示类与类之间的关系是整体与部分的关系。在需求描述中有"包含"、"组成"等词时，常常存在着聚集关系。聚集的符号是在关联的整体类端多了一个菱形框，聚集的表示方法如图 7-12 所示。

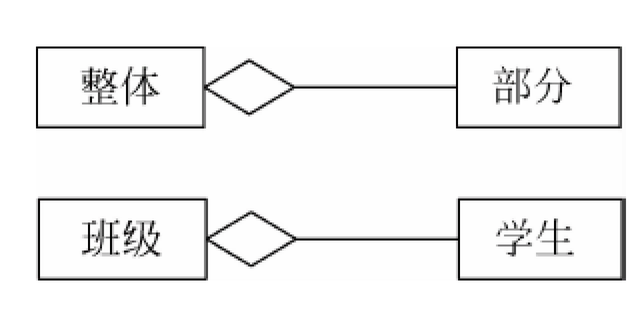


图 7-12 聚集的表示方法

一般化关系就是通常所说的继承关系，它是通用元素和具体元素之间的一种分类关系。一般化类又被称为父类，具体类又被称为子类，各子类继承了父类的性质，而各子类的一些共同性质和操作又被归纳到父类中。一般化关系的符号表

示是在类关联的连线上加上一个小三角形，表示方法如图 7-13 所示。

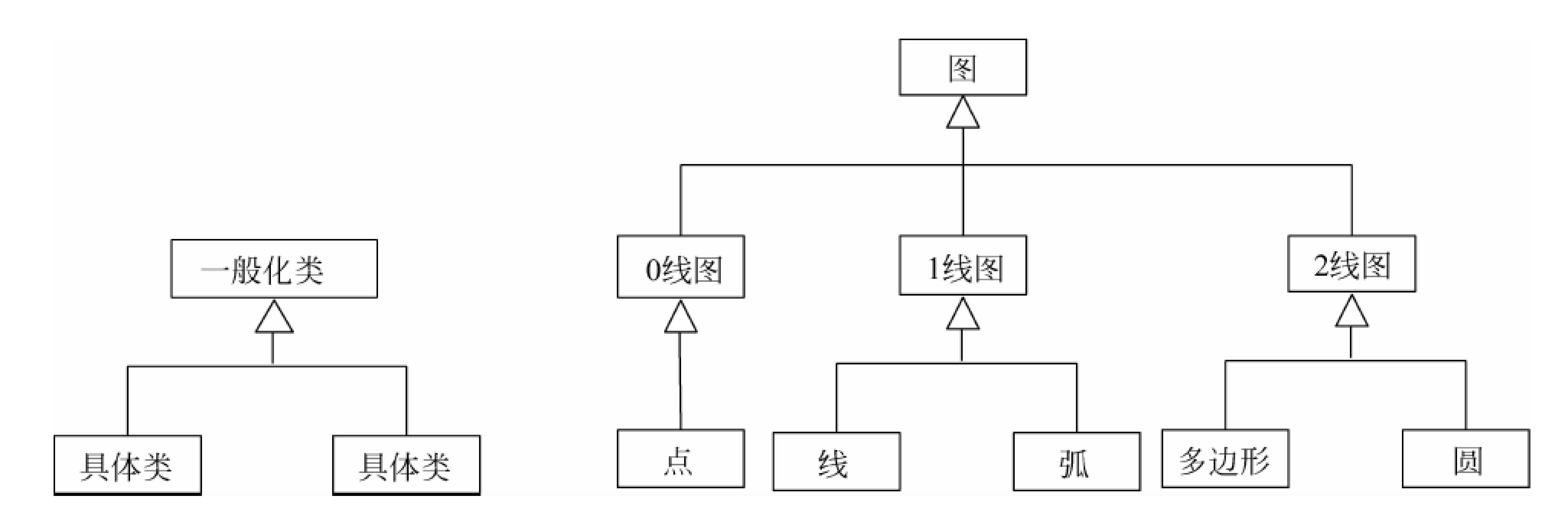


图 7-13　一般化关系的表示方法

2. 动态模型

动态模型表示瞬时的、行为化的、系统的控制性质，它规定了对象模型中对象的合法变化序列。它体现了系统的控制、操作的执行顺序，它从对象的事件和状态角度出发，表现了对象的相互行为。一旦建立起对象模型后，就需要考虑对象的动态行为，动态模型使用状态图作为描述工具，用于描述系统的状态和事件。状态图的表示方法如图 7-14 表示。

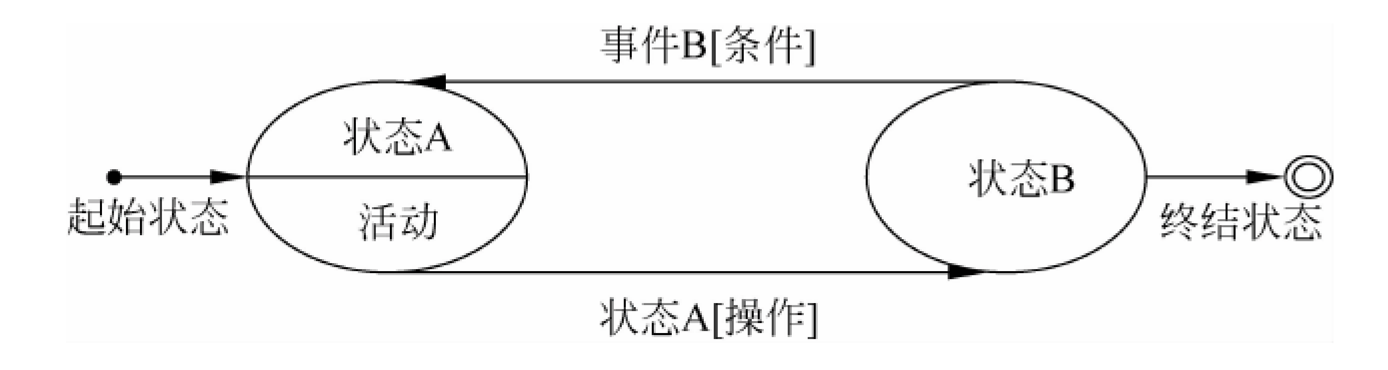


图 7-14　状态图的表示方法

3. 功能模型

功能模型表示变化的系统的功能性质，更直接地反映了用户对系统的要求。功能模型由一组数据流图组成，数据流图说明数据流是如何从外部输入、经过操作和内部存储输出到外部的。功能模型如图 7-15 所示。

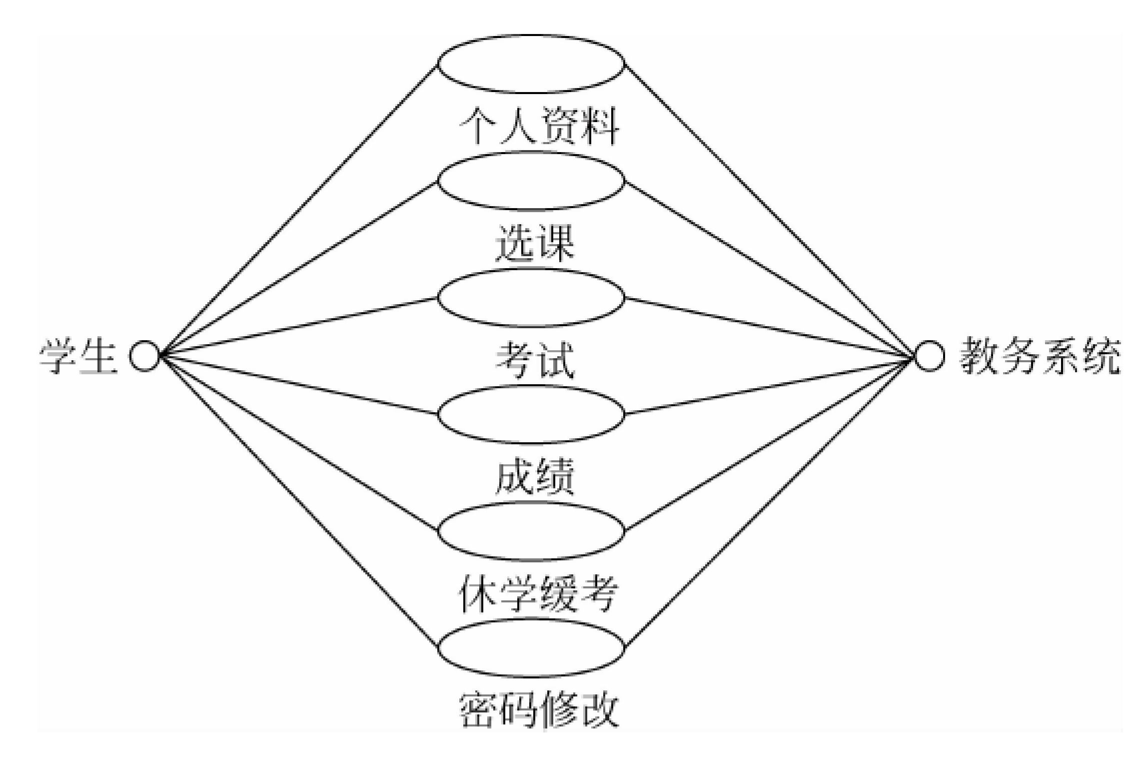


图 7-15　功能模型

综上所述，三种模型分别从三个不同侧面描述了所要开发的系统，这三种模型相互补充、相互配合，使我们对系统的认识更加全面。功能模型定义了系统应该“做什么”；动态模型定义了系统“何时做”；对象模型定义了“对谁做”。

7.4.4 设计类

类与对象是在问题域中客观存在的，系统分析员的主要任务就是通过分析找出这些类与对象。首先，以用户需求的陈述为依据，把陈述中的名词作为类或对象的候选者，形容词作为对象的属性，把动词作为操作的候选，通过分析找出候选的类与对象，然后严格考察每个候选对象，去掉无关、冗余的信息，明确描述对象的属性和操作，从候选的类与对象中筛选掉不正确的或不必要的类。

分析类与对象间存在的关联关系，如果两个或多个对象之间有相互依赖、相互作用的关系，它们就是关联的。大多数关联可以通过直接提取用户需求中的动词得到，通过仔细分析，还能发现一些陈述中隐含的关联。还需要对初步分析出的关联进行进一步筛选，以去掉不正确或不必要的关联，对确定下来的关联进行进一步完善。

属性是对象的性质，借助于属性人们能对类与对象和结构有更深入、更具体的认识。确定属性的过程包括分析和选择两个步骤，在分析过程中应该首先找出最重要的属性，以后再逐渐把其余属性添加进来，不要考虑那些纯粹用于实现的属性，应分析确定下来的属性，从中删除不必要的属性。

确定了类中应该定义的属性之后，应对系统中众多的类加以组织建立继承关系。经过反复修改逐步得到完善的模型。在实际工作中，并不一定按照前面讲述的次序开展工作，可以按照自己的习惯安排工作的次序，也可以初步完成一部分工作，再返回来加以完善。但如果是初次接触面向对象方法，最好使用本书中介绍的次序，尝试开发几个较小的系统，取得一些实际经验后，再总结出更适合自己的工作方式。

7.4.5 面向对象实现

面向对象实现是把面向对象设计结果翻译成用某种程序设计语言编写的面向对象程序，也就是常说的编码。作为软件工程的一个步骤，编码是设计的必然结果。在编写程序时，程序员都习惯使用自己常用的语言，由于现在的程序设计语言越来越多，开发软件系统时必须做出一个重要的抉择——使用什么样的程序设计语言实现系统。合适的程序设计语言能够使编码过程相对简单，减少程序的测试量，并且使得程序可读性好、易于维护。

那么，到底使用哪一种程序设计语言呢？现实中往往使用高级语言。高级语言有理想的模块化机制，可读性好的控制结构和数据结构。为了便于调试和提高软件可靠性，语言的编译程序应尽可能多地发现程序中的错误；为了降低软件的开发成本和维护成本，选用的语言应该有良好的独立编译机制。在实际选择语言时，往往要考虑以下九个因素。

(1) 语言自身的功能。根据不同系统的要求及语言自身特点来选择理想的开发语言。

(2) 系统用户的要求。所开发的系统由用户自己负责维护，用户通常要求选择他们熟悉的语言来编写程序。

(3) 编码和维护成本。合适的程序设计语言可大幅度降低程序的编码量，降低维护工作量，从而降低编码和维护成本。

(4) 可以使用的编译程序。运行目标系统的环境可以提供的编译程序往往限制了可以选用的语言范围。

(5) 可以使用的软件工具和组件。有些软件工具，如文本编辑器、编码控制系统组建库等，在支持程序的过程中起着重要作用。选用的程序设计语言是否可以使用这些工具，将影响系统实现和测试的工作量。

(6) 软件的兼容性。虽然高级语言的适应性很强，但不同机器上所配备的语言可能不同，子系统和主系统之间所采用的机器类型也可能不同，应充分考虑不同情况下软件的兼容性。

(7) 软件的可移植性。一般选择标准化程度高、程序可移植性好的程序设计语言，使所开发的软件将来能够移植到不同的硬件环境下运行。

(8) 系统规模。如果开发的系统庞大，而现有的语言不能够完全适用，就需要设计一种能供这个工程项目专用的程序设计语言。

(9) 程序员的知识。对于有经验的程序员来说，学习一种语言并不困难，但是要完全掌握一种新语言却需要实践。一般选用程序员熟练掌握的程序设计语言，以方便代码的编写。

好的程序设计语言有助于编写既可靠又容易维护的程序，但是工具再好，使用不当也不会达到预期的效果。良好的程序设计风格对编码来说尤为重要，好的编码风格不仅能明显减少维护量，而且有助于在新项目中重用已有的程序代码。应该用统一、标准的格式编写源程序。提高程序的可读性，常用的方法有以下八个：

(1) 标识符"见名知义"。有助于阅读者理解程序，如果使用缩写，缩写的规则应该一致。例如：学生用 stu 表示，学生数学成绩用 stu_mat 表示，计算机成绩的平均值用 ave_comp 表示。

(2) 添加必要的注释。注释是程序员和程序阅读者之间交流的桥梁，正确的注释有助于任何人对程序的理解，并为以后的测试和维护提供明确的指导信息。通常在每个模块开始处有一段序言性注释，简要描述模块的功能、主要算法、接口特点、重要数据，对理解后面的程序具有引导作用。功能性注释插在程序中间，是一段与程序代码有关的注释，用以描述以后的语句或程序段是在做什么工作。

(3) 在注释与程序段、不同程序段之间插入空行。

(4) 每行只写一条语句。

(5) 数据说明的次序应该标准化，说明的先后次序要固定。

(6) 使用标准的控制结构，有助于语句简单明了。

(7) 尽可能使用库函数。

(8) 利用括号等使逻辑表达式或算术表达式使运算次序清晰直观。

7.5 面向元数据设计

元数据为描述数据的数据(Data About Data)，主要是描述数据属性(Property)的信息，用来支持如指示存储位置、历史数据、资源查找、文件记录等功能。元数据算是一种电子式

目录，为了达到编制目录的目的，必须可描述并收藏数据的内容或特色，进而达成协助数据检索的目的。

元数据是对信息资源的规范化描述，它是按照一定标准，从信息资源中抽取出相应的特征，所组成的一个特征元素结合。这种规范化描述可以准确和完备地说明信息资源的各项特征，不同类型的数据资源可能会有不同的元数据标准。元数据内容标准一般包括了描述一个具体对象时所需要的数据项集合、各数据项语义定义、计算机应用时的语法规定等。

元数据为信息的管理、发现和获取提供一种实际而简便的方法。通过元数据，人们能够对信息资源进行详细、深入地了解，包括信息资源的格式、质量、处理方法、获取方法等各方面的细节，对于数据生产者来说可以利用元数据进行数据维护、历史资料维护等工作。

元数据可用于描述要素、数据集或数据集系列的内容、覆盖范围、质量、管理方式、数据的所有者、数据的提供方式等有关的信息。元数据以非特定语言的方式描述在代码中定义的每一类型和成员。元数据可存储以下信息：

- 程序集的说明。
- 标识（名称、版本、区域性和公钥）。
- 导出的类型。
- 该程序集所依赖的其他程序集。
- 运行所需要的安全权限。
- 类型的说明。
- 名称、可见性、基类和实现的接口。
- 成员，如方法、字段、属性、事件、嵌套的类型。
- 属性。
- 修饰类型和成员的其他说明性元素。

元数据结构可分为：内容结构（Content Structure）、句法结构（Syntax Structure）和语义结构（Semantic Structure）。

- 内容结构：定义元数据的构成元素，包括描述性元素、技术性元素、管理性元素和结构性元素。这些数据元素很可能依据一定标准来选取，因此在元数据内容结构中需要对此进行说明。
- 句法结构：定义格式结构及其描述方式，例如元素的分区分段组织、元素选取使用规则、元素描述方法、元素结构描述方法、结构语句描述语言等。有时，句法结构需要指出元数据是否与所描述的数据对象捆绑在一起，或作为单独数据存在，但以一定形式与数据对象链接。
- 语义结构：定义元素的具体描述方法。例如，描述元素时所采用的标准、最佳实践或自定义的描述要求。

元数据的应用主要有以下四个方面：

（1）确认和检索（Confirm and Searching）。主要致力于如何帮助人们确认和检索所需要的资源，数据元素往往限于作者、标题、主题、位置等简单信息。

（2）著录描述（Cataloging）。用于对数据单元进行详细、全面地著录描述，数据元素包括内容、载体、位置与获取方式、制作与利用方法、甚至相关数据单元等方面。

（3）资源管理（Resource Administration）。支持资源的存储和使用管理，数据元素除比

较全面的著录描述信息外，还往往包括权利管理、电子签名、资源评鉴、使用管理、支付审计等方面的信息。

(4) 资源保护与长期保存(Preservation and Archiving)。支持对资源进行长期保存，数据元素除对资源进行描述和确认外，往往包括详细的格式信息、制作信息、保护条件、转换方式、保存责任等内容。

元数据编码语言指对元数据元素和结构进行定义和描述的具体语法和语义规则，常被称为定义描述语言 DDL。在元数据发展初期人们常使用自定义的记录语言(如 MARC)或数据库记录结构(如 ROADS)，但随着元数据格式的增多和互操作的要求，人们开始采用一些标准化的 DDL 来描述元数据，例如 SGML 和 XML，其中 XML 最有潜力。

7.6 软件设计方法总结

设计软件首先要明确软件设计的目标和任务，根据软件的功能需求进行数据设计、系统结构设计和过程设计。其中数据设计侧重于数据结构的定义；系统结构设计用于定义软件系统各主要成分之间的关系；过程设计则是把结构成分转换成软件的过程性描述，在编码步骤中，根据这种过程性描述，生成源程序代码，然后通过测试最终得到完整有效的软件。

从工程管理角度看，软件设计分两步完成。

(1) 概要设计：将软件需求转化为数据结构和软件的系统结构。

(2) 详细设计：即过程设计，通过对结构表示进行细化，得到软件的详细的数据结构和算法。

在进入软件开发阶段之初，首先应为软件开发组制定在设计时应该共同遵守的标准，以便协调组内各成员的工作，包括以下内容：

- 阅读和理解软件需求说明书，确认能否实现用户要求，明确实现的条件，从而确定设计的目标以及它们的优先顺序。
- 根据目标确定最合适的设计方法。
- 规定设计文档的编制标准。
- 规定编码的信息形式，与硬件、操作系统的接口规约，命名规则。

基于软件功能层次结构建立系统，采用某种设计方法，将系统按功能划分成模块的层次结构、确定每个模块的功能、建立与已确定的软件需求的对应关系、确定模块间的调用关系、模块间的接口、评估模块划分的质量。

在数据结构设计中，确定软件涉及的文件系统的结构以及数据库的模式和子模式，进行数据完整性和安全性的设计；确定输入、输出文件的详细的数据结构；结合算法设计，确定算法所必须要的逻辑数据结构及其操作；确定对逻辑数据结构所必需的那些操作的程序模块；限制和确定各个数据设计决策的影响范围；数据的保护性设计等。

可靠性设计也叫质量设计，在运行过程中，为了适应环境的变化和用户新的要求，需要经常对软件进行改造和修正。在软件开发的一开始就要确定软件可靠性和其他质量指标，考虑相应措施，以使得软件易于修改和易于维护。

概要设计评审阶段需要检查设计的可追溯性、接口、风险、实用性、技术清晰度、可维护

性、质量、各种选择方案、限制等方面。

在详细设计过程中，需要完成的工作有：

- 确定软件各个组成部分内的算法以及各部分的内部数据组织。
- 选择某种过程的表达形式来描述各种算法。
- 评审详细设计。

7.7 软件设计文档

在整个软件设计过程中，文档的编写和保存也是非常重要的，规范的文档可方便下一阶段工作的开展，明确阶段任务，保证软件开发的顺利进行。文档是影响软件可维护性的决定因素，由于长期使用的大型软件系统在使用过程中必然会经过多次修改，所以文档比程序代码更重要。软件系统的文档可分为用户文档和系统文档，用户文档主要描述系统功能和使用方法，并不关心这些功能是怎样实现的；系统文档描述系统设计、实现和测试等各方面的内容。

1. 用户文档

用户文档是用户了解系统的窗口，它使用户获得对系统准确的初步印象，文档的结构方式应该使用户能够方便地根据需要阅读有关的内容。用户文档至少应该包括以下几方面内容。

- 功能描述：说明系统能够做什么。
- 安装文档：阐述如何安装系统、怎样配置硬件。
- 使用手册：简要介绍如何使用这个系统。
- 参考手册：详细描述所有系统设施以及它们的使用方法，还应该解释系统可能产生的各种出错信息的含义。
- 操作员指南：说明操作员应该如何处理在使用中出现的各种情况。

2. 系统文档

系统文档是指从问题定义、需求说明到验收测试计划一系列和系统实现有关的文档。描述系统设计、实现和测试的文档对于理解程序和维护程序来说是至关重要的。通过系统文档使读者对系统的框架有一个全方位的认识，引导读者对系统每个方面每个特点进行具体深入地认识。

在整个软件开发过程中，也会产生大量的文档资料，例如概要设计完成时应编写以下文档：概要设计说明书、数据库设计说明书、用户手册、初步的测试计划。其中，数据库设计说明书包括引言、引用文件、数据库级设计决策、数据库详细设计、用于数据库操纵或访问的软件配置项的详细设计、需求的可追踪性、注解、附录等。

这些文档都是开发过程的宝贵资料，保存好这些文档对于日后代码的修改和维护及其重要。

7.8 本章小结

在软件设计阶段可考虑系统各种可能的实现方案，并从中选出最佳方案，确定每个功能由哪些模块组成，这些模块间的关系，确定每个模块的处理过程。

软件设计是后续开发步骤及软件维护工作的基础。如果没有好的设计，则只能建立一个不稳定的系统结构。在软件分析阶段可确定需要做什么和系统需求规格，在软件开发阶段则确定了系统如何实现。本章详细讲解软件设计的整个过程、软件设计原理，以及简要说明了面向过程设计和面向对象设计。

习题 7

1. 什么是模块？什么是模块化？
2. 什么是抽象？为什么在软件结构化设计中要用到抽象？
3. 请给出逐步求精的定义。抽象与逐步求精之间的关系是什么？
4. 衡量模块独立性的两个标准是什么？它们各表示什么含义？
5. 模块间的耦合有哪几种？它们各表示什么含义？
6. 模块间的内聚有哪几种？它们各表示什么含义？
7. 什么是对象，它的构成要素有哪些？分别阐述这些要素的概念。
8. 什么是类，它与对象的关系是什么？
9. 在面向对象方法学中，为什么要用继承手段？它的主要作用是什么？
10. 简述封装的含义和封装机制的作用。
11. 解释什么是对象模型、动态模型和功能模型。
12. 一本书有一个封面、一个目录、一个前言、若干章，每章有若干节，每节有若干个段，每段有若干个句子，每节有 0 个或多个图或表，最后还有一个封底。根据以上描述建立该书的对象模型。
13. 选择面向对象程序设计语言时主要应该考虑哪些因素？良好的面向对象程序设计风格主要有哪些准则？

第8章 软件测试

编码结束后，开始进入测试阶段。无论采用何种模型开发出来的系统，在设计中都有可能存在错误或漏洞，在编码过程中也可能会引入新的错误，所以在软件交付使用之前，必须进行严格测试，通过测试找出软件在需求分析、设计和编码阶段隐藏的错误，并加以改正。

由于软件产品具有逻辑复杂性，所以软件测试的工作量和工作难度不亚于软件分析和设计。据统计，测试工作量占软件开发总工作量的40%～50%以上，而测试的范围存在于软件的整个生命周期，不仅仅局限在程序编码阶段。

8.1 软件测试概论

为了发现程序中的错误而执行程序的过程被称为测试。软件开发的前几个阶段是构建软件系统，而软件测试的目的则是尽力找出软件的失败和不足之处。从表面上看，设计是建设性的，测试是破坏性的。事实上这两个过程都是为了提高软件的质量。测试是保证软件质量的重要手段之一。

8.1.1 测试的目的

软件测试是对软件计划、软件设计和软件编码进行纠错的活动，测试的目的是找出整个软件开发周期中各个阶段的错误，分析错误的性质和位置并加以纠正。纠正的过程包括对文档和代码的修改，找错的活动被称为测试，而纠错的过程被称为调试。

正确地认识测试十分重要，如果为了表明程序是正确的而进行测试，就会设计出一些不易暴露错误的测试方案；相反，如果测试是为了发现程序中的错误，就会力求设计出最能暴露错误的测试方案。

测试的目的决定了测试方案的设计，G. Myers给出了软件测试的目的：

- 测试是为了发现程序中的错误而执行程序的过程。
- 好的测试方案是极有可能发现迄今尚未发现的尽可能多的错误的测试。
- 成功的测试是发现了迄今尚未发现的错误的测试。

怎样才能实现测试目的呢？为了设计出有效的测试方案，软件工程师必须深入理解并正确运用指导软件测试的基本准则。这些基本准则有：

- 所有测试都应该能追溯到用户需求。
- 应该在测试之前就制定出测试计划。

- 应该从“小规模”测试开始，逐步进行“大规模”测试。
- 穷举测试是不可能的。
- 为了达到测试效果，应该由独立的第三方从事测试工作。

8.1.2 测试的基本原则

为了达到测试的要求，测试人员在进行程序测试时，应该遵循一些测试原则：

- 在测试前要认定被测试的软件有错，不要认为被测试的程序是正确的。
- 尽量避免测试自己编写的程序。
- 测试时要考虑合理的输入数据和不合理的输入数据。
- 测试时应以软件需求规格说明书中的需求为标准。
- 要确定找到的新错与已找到的旧错成正比。G. Myers 认为“一个或多个模块中存在错误的概率与其中已经发现的错误个数成正比”，因此，应该对已经发现错误集中的模块进行重点测试，以找出相关的可能错误，提高测试效率。
- 所有的测试用例，都应该被记录下来，以供后来的测试和维护使用。

8.2 软件测试模型

1. V 模型

在软件测试方面，V 模型是最广为人知的模型，尽管很多富有实际经验的测试人员还是不太熟悉 V 模型。V 模型已存在了很长时间，和瀑布开发模型有着一些共同的特征，由此也和瀑布模型一样的受到了批评和质疑。V 模型中的过程从左到右，描述了基本的开发过程和测试行为。

V 模型的优点在于它非常明确地标明了测试过程中存在的不同级别，并且清楚地描述了这些测试阶段和开发过程期间各阶段的对应关系。V 模型的缺点在于它把测试作为编码之后的最后一个活动，需求分析等前期产生的错误直到后期的验收测试时才能被发现。V 模型如图 8-1 所示。

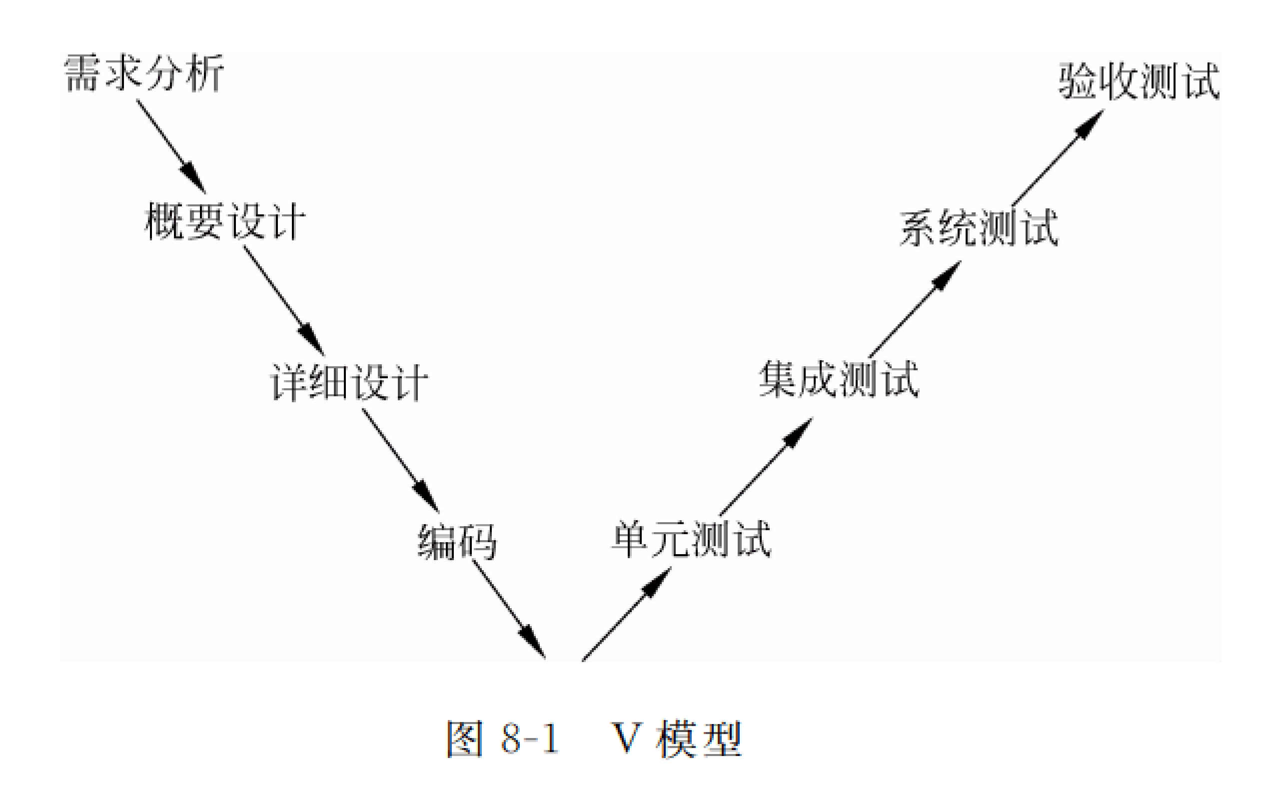


图 8-1 V 模型

2. W模型

V模型的局限性在于没有明确地说明早期的测试，无法体现“尽早地和不断地进行软件测试”的原则。在V模型中增加软件各开发阶段应同步进行的测试，则V模型就演化为了W模型。在模型中不难看出，开发是V，测试是与此并行的V。基于“尽早地和不断地进行软件测试”的原则，在软件的需求和设计阶段的测试活动应遵循IEEE 1012-1998《软件验证与确认(V&V)》的原则。

W模型由Evolutif公司提出，相对于V模型，W模型更科学。W模型是V模型的发展，强调的是测试伴随着整个软件开发周期，而且测试的对象不仅仅是程序，需求、功能和设计同样要被测试。测试与开发是同步进行的，从而有利于尽早地发现问题。图8-2为在V模型中增加同步测试的方法。

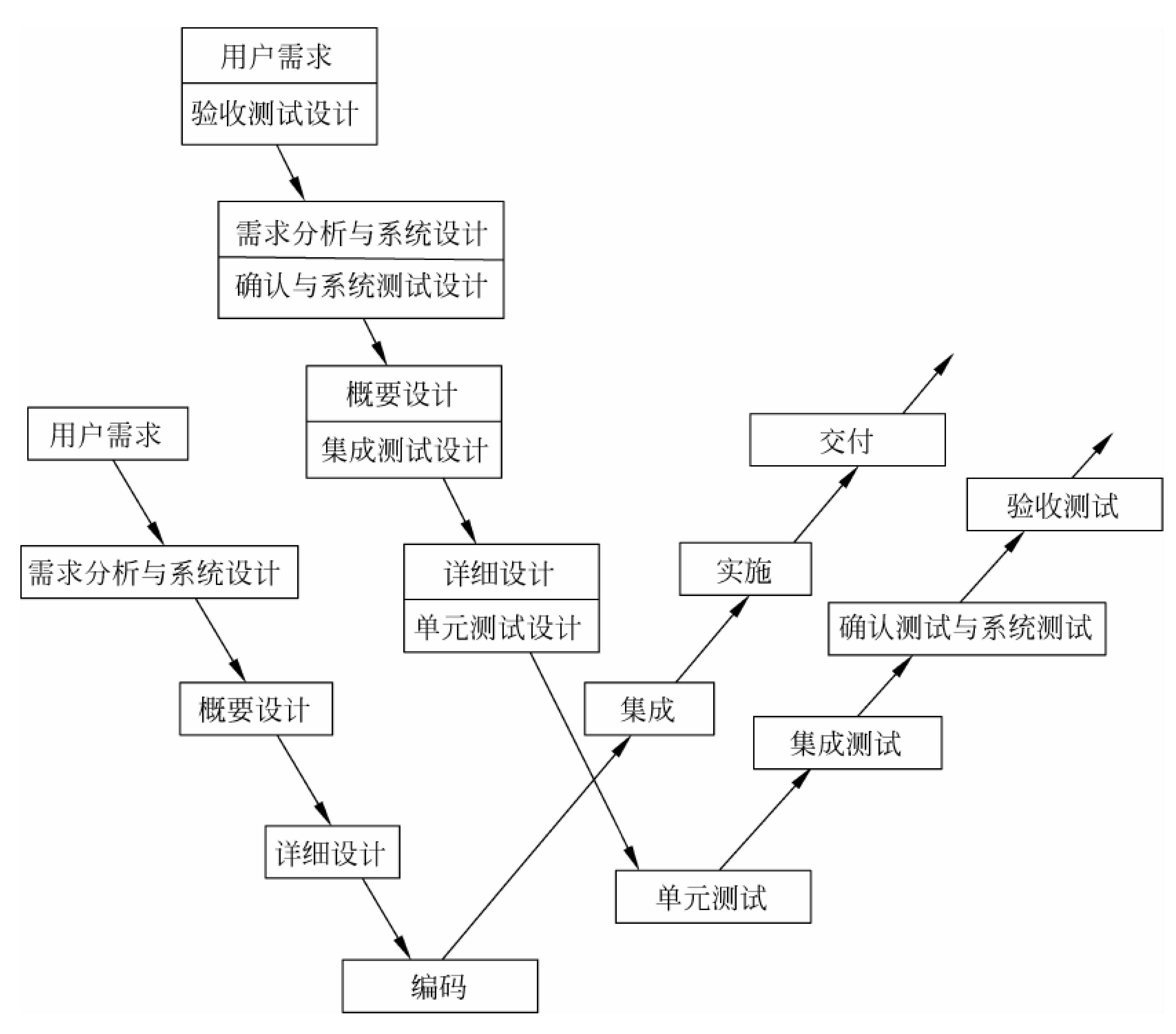


图8-2 在V模型中增加同步测试的方法

W模型也有局限性。W模型和V模型都把软件的开发视为需求、设计、编码等一系列串行的活动，无法支持迭代、自发性以及变更调整。W模型如图8-3所示。

3. X模型

X模型也是对V模型的改进，X模型提出针对单独的程序片段进行相互分离的编码和测试，此后通过频繁地交接，通过集成最终合成为可执行的程序。

X模型的左边描述的是针对单独程序片段所进行的相互分离的编码和测试，此后将进行频繁地交接，通过集成最终成为可执行的程序，然后再对这些可执行程序进行测试。已通

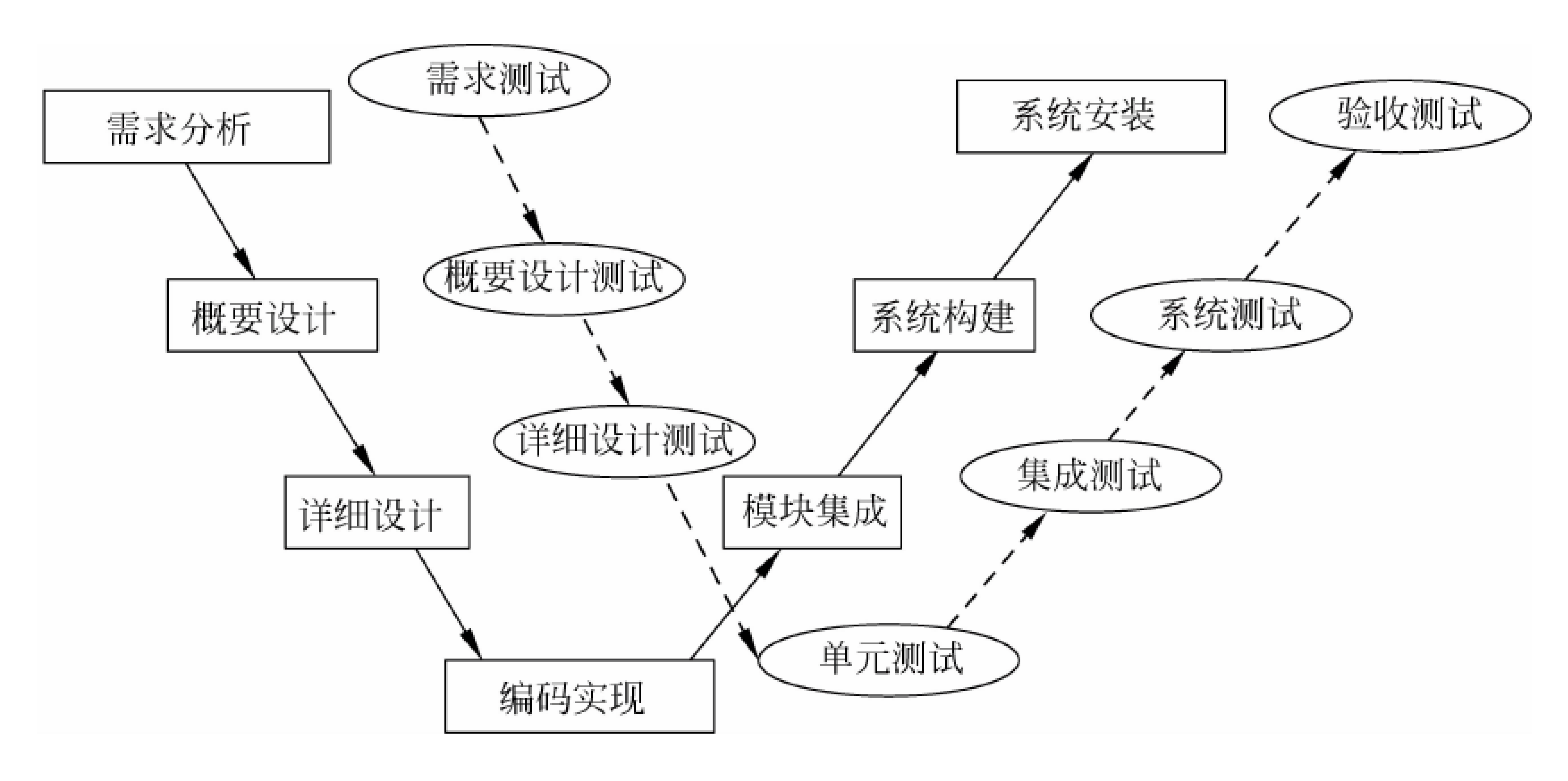


图 8-3 W 模型

过集成测试的成品可以进行封装并提交给用户，也可以作为更大规模和范围内集成的一部分。多根并行的曲线表示变更可以在各个部分发生。由图 8-4 中可见，X 模型还定位了探索性测试，这是不进行事先计划的特殊类型的测试，这一方式往往能帮助有经验的测试人员在测试计划之外发现更多的软件错误。但这样可能对测试造成人力、物力和财力的浪费，对测试员的熟练程度要求比较高。

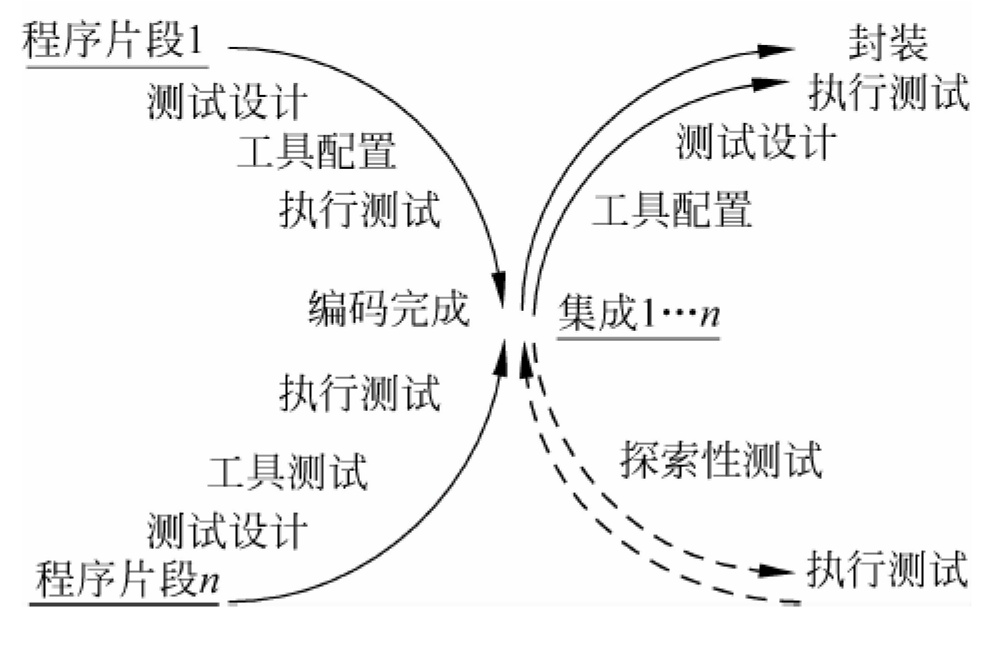


图 8-4 X 模型

4. H 模型

在 H 模型中，软件测试过程活动完全独立，贯穿于整个产品周期，与其他流程并发地进行，当某个测试点准备就绪时，就可以从测试准备阶段进入测试执行阶段。软件测试可以尽早地进行，并且可以根据被测物的不同而分层次进行。

图 8-5 演示了在整个生产周期中某个层次上的一次测试的“微循环”。图中标注的“其他流程”可以是任意的开发流程，例如设计流程或者编码流程。也就是说，只要测试条件成熟了，测试准备活动完成了，测试执行活动就可以进行了。

H 模型揭示了一个原理：软件测试是一个独立的流程，贯穿产品整个生命周期，与其他流程并发地进行。H 模型指出软件测试要尽早准备，尽早执行。不同的测试活动可以是按照某个次序先后进行的，但也可能是反复进行的，只要某个测试达到准备就绪点，测试执行活动就可以展开。

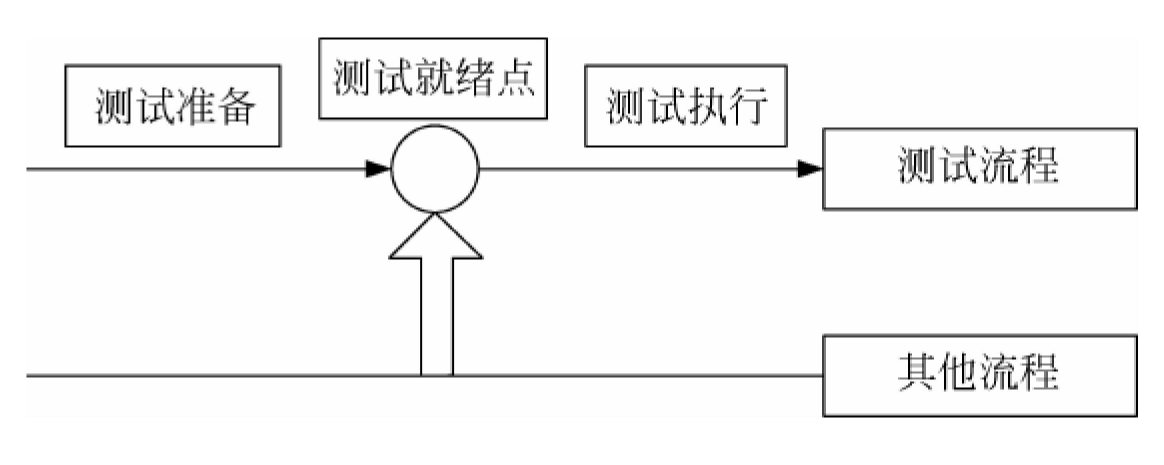


图 8-5　H 模型

5. 前置测试模型

前置测试模型是由 Robin Fgoldsmith 等人提出的，是一个将测试和开发紧密结合的模型，该模型提供了轻松的方式，可以加快项目的实现速度。前置测试模型可参考图 8-6。

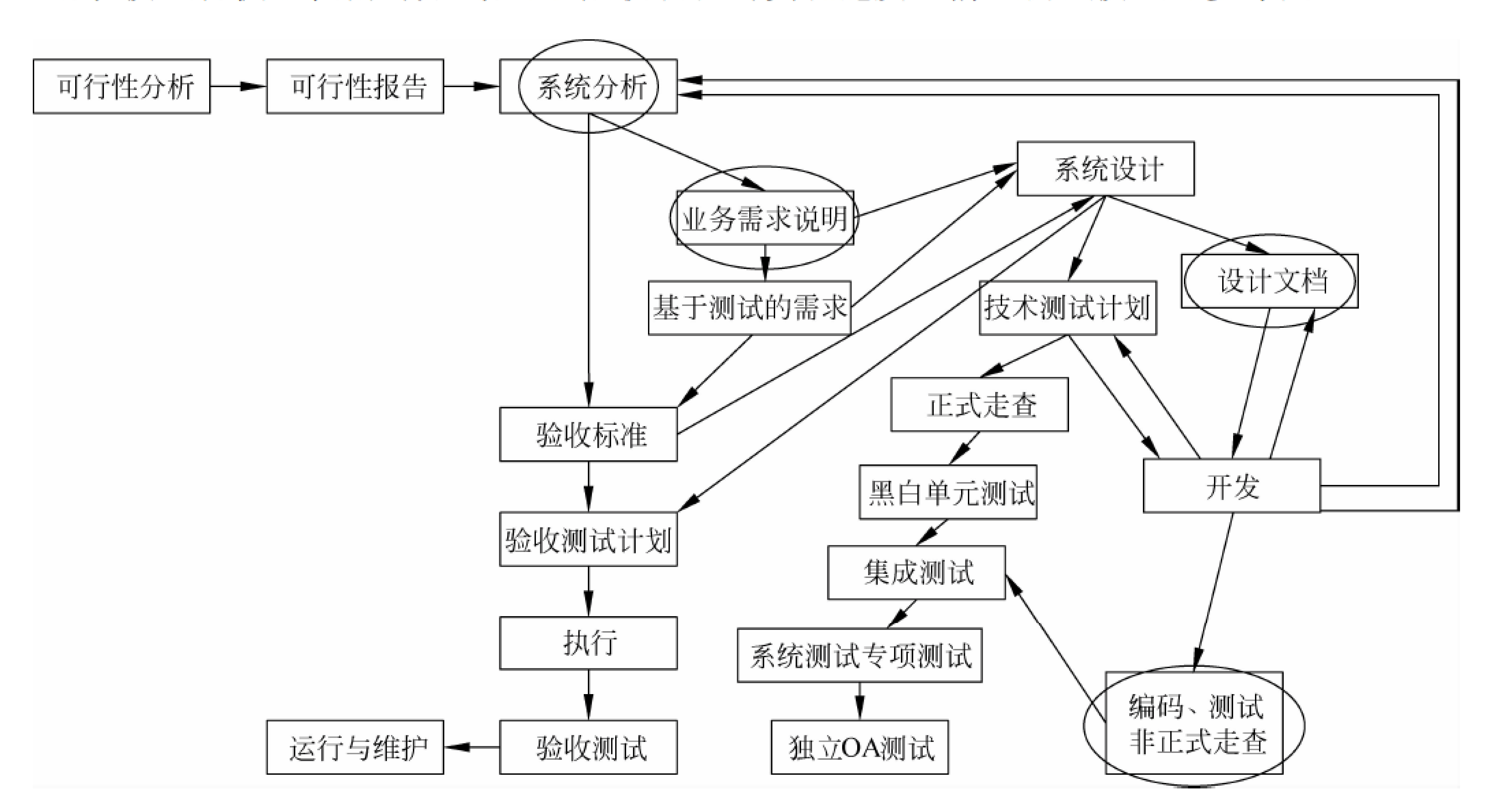


图 8-6　前置测试模型

前置测试的特点有以下几个：

(1) 开发和测试相结合。前置测试模型将开发和测试的生命周期整合在一起，标识了项目生命周期从开始到结束之间的关键行为。并且表示了这些行为在项目周期中的价值所在。如果其中有些行为没有得到很好地执行，那么项目成功的可能性就会因此降低。如果有业务需求，则系统开发过程将更有效率。在没有业务需求的情况下进行开发和测试是不可能的。而且，业务需求最好在设计和开发之前就被正确定义的。

(2) 对每一个交付内容进行测试。每一个交付的开发结果都必须通过一定的方式进行测试，源程序代码并不是唯一需要测试的内容。

(3) 在设计阶段进行计划和测试设计。设计阶段是做测试计划和测试设计的最好时机。

(4) 测试和开发结合在一起。前置测试将测试执行和开发结合在一起，并在开发阶段以编码-测试-编码-测试的方式来体现。也就是说，程序片段一旦编写完成，就会立即进行测

试。在一般情况下，先进行的测试是单元测试，因为开发人员认为通过测试来发现错误是最经济的方式。但也可参考 X 模型，即一个程序片段也需要相关的集成测试，甚至有时还需要一些特殊测试。对于一个特定的程序片段，其测试的顺序可以按照 V 模型的规定，但其中还会交织一些程序片段的开发，而不是按阶段完全地隔离。

(5) 让验收测试和技术测试保持相互独立。验收测试应该独立于技术测试，这样可以提供双重的保险，以保证设计及程序编码能够符合最终用户的需求。验收测试既可以在实施阶段的第一步执行，也可以在开发阶段的最后一步执行。

(6) 反复交替的开发和测试。在项目中从很多方面都可以看到变更的发生，例如需要重新访问前一阶段的内容，或者跟踪并纠正以前提交的内容，修复错误，排除多余的成分，以及增加新发现的功能等。开发和测试需要一起反复交替地执行。模型并没有明确指出参与的系统部分的大小。这一点和 V 模型中所提供的内容相似。不同的是，前置测试模型对反复和交替进行了非常明确描述。

(7) 发现内在的价值。前置测试能给需要使用测试技术的开发人员、测试人员、项目经理、用户等带来很多不同于传统方法的内在的价值。与以前的方法中很少划分优先级所不同的是，前置测试用较低的成本及早地发现错误，并且充分强调了测试对确保系统高质量的重要意义。前置测试代表了对测试的、新的、不同的观念。在整个开发过程中，反复使用了各种测试技术以使开发人员、经理和用户节省各自的时间，简化各自的工作。

8.3 黑盒测试方法

测试任何产品都有两种方法：如果已经知道了产品应该具有的功能，可以通过测试来检验是否每个功能都能正常使用；如果知道产品的内部工作过程，可以通过测试来检验产品内部工作是否按照规格说明书的规定正常进行。前一种方法称为黑盒测试，后一种方法称为白盒测试。

黑盒测试也称功能测试，它是通过测试来检测每个功能是否都能正常使用。在测试中，把程序看做一个不能打开的黑盒子，在完全不考虑程序内部结构和内部特性的情况下，在程序接口进行测试，它只检查程序功能是否按照需求规格说明书的规定正常使用，程序是否能适当地接收输入数据而产生正确的输出信息。黑盒测试着眼于程序外部结构，不考虑内部逻辑结构，主要针对软件界面和软件功能进行测试。

具体的黑盒测试用例设计方法包括等价类划分法、边界值分析法、错误推测法、因果图法、判定表驱动法、正交试验设计法、功能图法、场景法等。

8.3.1 等价类划分法

等价类划分法是把程序的输入域划分成若干部分(子集)，然后从每个部分中选取少数代表性数据作为测试用例。每一类的代表性数据在测试中的作用等价于这一类中的其他值。

等价类划分可有两种不同的情况：有效等价类和无效等价类。有效等价类是指由对于程序的规格说明来说是合理的、有意义的输入数据构成的集合。利用有效等价类可检验程

序是否实现了规格说明中所规定的功能和性能。无效等价类与有效等价类的定义恰巧相反。设计测试用例时，要同时考虑这两种等价类。因为，软件不仅要能接收合理的数据，也要能经受意外的考验。进行这样的测试才能确保软件具有更高的可靠性。划分等价类的原则有以下几个：

（1）在输入条件规定了取值范围或值的个数的情况下，则可以确立一个有效等价类和两个无效等价类。

（2）在输入条件规定了输入值的集合或者规定了“必须如何”的条件的情况下，可确立一个有效等价类和一个无效等价类。

（3）在输入条件是一个布尔量的情况下，可确定一个有效等价类和一个无效等价类。

（4）在规定了输入数据的一组值（假定 n 个），并且程序要对每一个输入值分别处理的情况下，可确立 n 个有效等价类和一个无效等价类。

（5）在规定了输入数据必须遵守的规则的情况下，可确立一个有效等价类（符合规则）和若干个无效等价类（从不同角度违反规则）。

（6）在确认已划分的等价类中各元素在程序处理中的方式不同的情况下，则应再将该等价类进一步划分为更小的等价类。

对划分出的等价类可按以下三个原则设计测试用例：

（1）为每一个等价类规定一个唯一的编号。

（2）设计一个新的测试用例，使其尽可能多地覆盖尚未被覆盖的有效等价类。重复这一步，直到所有的有效等价类都被覆盖为止。

（3）设计一个新的测试用例，使其仅覆盖一个尚未被覆盖的无效等价类。重复这一步，直到所有的无效等价类都被覆盖为止。

8.3.2 边界值分析法

长期的测试工作经验证明，大量的错误是发生在输入或输出范围的边界上，而不是发生在输入输出范围的内部。因此针对各种边界情况设计测试用例，可以查出更多的错误。边界值分析法是对等价类划分方法的补充，边界值分析法既重视输入条件边界，而且也重视输出域边界。

使用边界值分析方法设计测试用例，首先应确定边界情况。通常输入和输出等价类的边界，就是应着重测试的边界情况。应当选取正好等于、刚刚大于或刚刚小于边界值的值作为测试数据，而不是选取等价类中的典型值或任意值作为测试数据。

基于边界值分析方法选择测试用例的原则有以下几个：

（1）如果输入条件规定了值的范围，则应选取刚达到这个范围的边界的值，以及刚刚超出这个范围边界的值作为测试输入数据。

（2）如果输入条件规定了值的个数，则用最大个数、最小个数、比最小个数少一、比最大个数多一的数作为测试数据。

（3）如果程序的规格说明给出的输入域或输出域是有序集合，则应选取集合的第一个元素和最后一个元素作为测试用例。

（4）如果程序中使用了一个内部数据结构，则应当选择这个内部数据结构的边界上的值作为测试用例。

（5）分析规格说明，找出其他可能的边界条件。

8.3.3 错误推测法

错误推测法是基于经验和直觉推测程序中所有可能存在的错误，从而有针对性地设计测试用例的方法。

错误推测法的基本思想：列举出程序中所有可能有错误的和容易发生错误的特殊情况，根据它们选择测试用例。例如，在单元测试时曾列出的许多在模块中常见的错误、以前产品测试中曾经发现的错误等。还有，输入数据和输出数据为0的情况，输入表格为空或输入表格只有一行。这些都是容易发生错误的情况，可选择这些情况下的例子作为测试用例。

8.3.4 因果图法

前面介绍的等价类划分法和边界值分析法，都是着重考虑输入条件，但未考虑输入条件之间的联系、相互组合等。考虑输入条件之间的相互组合，可能会产生一些新的情况。但要检查输入条件的组合不是一件容易的事情，即使把所有输入条件划分成等价类，它们之间的组合情况也相当多。因此必须考虑采用一种适合于描述对于多种条件的组合，相应产生多个动作的形式来设计测试用例，这就需要利用因果图（也被称为逻辑模型）。

因果图方法最终生成的就是判定表，它适合于检查程序输入条件的各种组合情况。用因果图法生成测试用例的步骤分为以下五个：

（1）分析软件规格说明描述，哪些是原因（即输入条件或输入条件的等价类）、哪些是结果（即输出条件），并给每个原因和结果赋予一个标识符。

（2）分析软件规格说明描述中的语义。找出原因与结果之间、原因与原因之间对应的关系，根据这些关系画出因果图。

（3）由于语法或环境限制，有些原因与原因之间、原因与结果之间的组合情况是不可能出现的。为表明这些特殊情况，应在因果图上用一些记号标明约束条件或限制条件。

（4）把因果图转换为判定表。

（5）把判定表的每一列拿出来作为依据，设计测试用例。

从因果图生成的测试用例包括了所有输入数据取 True 与取 False 的情况，构成的测试用例数目达到最少，且测试用例数目随输入数据数目的增加而线性地增加。

8.4 白盒测试方法

白盒测试也称结构测试或逻辑驱动测试，它是按照程序内部的结构测试程序，通过测试来检测产品内部动作是否按照设计规格说明书的规定正常进行，检验程序中的每条通路是否都能按预定要求正确工作。这一方法是把测试对象看做一个打开的盒子，测试人员依据程序内部逻辑结构相关信息，设计或选择测试用例，对程序所有逻辑路径进行测试，通过在不同点检查程序的状态，确定实际的状态是否与预期的状态一致。

8.4.1 逻辑覆盖

逻辑覆盖包括语句覆盖、判定覆盖、条件覆盖、判定/条件覆盖、条件组合覆盖和路径覆盖。

1. 语句覆盖

语句覆盖(Statement Coverage)就是设计若干测试用例,运行要测试的程序,使得每一条可执行的语句至少执行一次。这是最常用的、也是最常见的一种覆盖方式,就是度量被测试代码中每个可执行语句是否都被执行到了。

使用语句覆盖的一个规则是设计的测试用例越少越好。语句覆盖率的公式可以表示如下:

语句覆盖率=被评价到的语句数量/可执行的语句总数×100%

语句覆盖常常被人指责为“最弱的覆盖”,它只管覆盖代码中的执行语句,却不考虑各种分支的组合等。假如只要求达到语句覆盖,那么换来的实际测试效果不明显,很难更多地发现代码中的问题。这里举一个简单的例子。

```
int fun(int a, int b)
{
return a / b;
}
```

假如我们的测试人员编写如下测试案例:

```
TeseCase: a = 10, b = 5
```

测试人员的测试结果会告诉你,他的代码覆盖率达到了100%,并且所有测试案例都通过了。然而遗憾的是,我们的语句覆盖率达到了所谓的100%,但是却没有发现最简单的bug。比如,当b=0时,会抛出一个除零异常。

2. 判定覆盖

判定覆盖又称为分支覆盖,就是设计若干个测试用例,运行被测程序,使得程序中每判定的取真分支和取假分支至少被评价一次。

判定覆盖的优点是判定覆盖具有比语句覆盖更强的测试能力,而且具有和语句覆盖一样的简单性,不需要细分每个判定就可以得到测试用例。判定覆盖的缺点是往往大部分的判定语句是由多个逻辑条件组合而成(如,判定语句中包含AND、OR、CASE),若仅仅判断其整个最终结果,而忽略每个条件的取值情况,必然会遗漏部分测试路径。例如下面的程序段:

```
int a,b;
if(a || b)
执行语句 1
else
执行语句 2
```

要达到这段程序的判断覆盖,应设计测试用例:

```
a = true, b = true
a = false, b = false
```

3. 条件覆盖

条件覆盖是指选择足够的测试用例,使得运行这些测试用例后,要使每个判断中每个条件的可能取值至少满足一次。条件覆盖要检查每个符合谓词的子表达式值为“真”和“假”两种情况,要独立衡量每个子表达式的结果,以确保每个子表达式的值为“真”和“假”两种情况都被测试到。以上例来说,要达到条件覆盖,应设计测试用例为:

```
a = true,b = true
a = true,b = false
a = false,b = true
a = false,b = false
```

条件覆盖通常比判定覆盖强,因为它使判定表达式中每个条件都取到了两个不同的结果,判定覆盖却只关心整个判定表达式的值。

4. 判定/条件覆盖

单独使用判定覆盖和条件覆盖,测试结果都不够全面,如果将两种覆盖结合,就会起到互相补充的作用,这就是判定/条件覆盖。判定/条件覆盖就是设计足够的测试用例,使得判断中每个条件的所有可能取值至少被执行一次,同时,每个判断的所有可能判定结果也至少被执行一次。

5. 条件组合覆盖

在白盒测试法中,应选择足够的测试用例,使得每个判定中条件的各种可能组合都至少出现一次。显然,满足“条件组合覆盖”的测试用例是一定满足“判定覆盖”、“条件覆盖”和“判定/条件覆盖”的。

6. 路径覆盖

在白盒测试法中,覆盖程度最高的就是路径覆盖,因为其覆盖程序中所有可能的路径。对于比较简单的小程序来说,实现路径覆盖是可能的,但是如果程序中出现了多个判断和多个循环,可能的路径数目将会急剧增长,以致实现路径覆盖是几乎不可能的。

通过路径分析,计算程序中的路径数(复杂度)。在下面的公式中,e 为边数,n 为结点数。

$$V(G) = e - n + 2$$

8.4.2 基本路径测试

利用基本路径测试方法设计测试用例是在给出程序控制流的基础上,分析控制结构的环路复杂性,导出基本的可执行路径,并把覆盖的路径压缩到一定范围内,保证程序中的循

环体最多只执行一次。设计出的测试用例要保证在测试中，程序中的每一条语句至少要被执行一次，而且每个条件在执行时都将分别取“真”、“假”两种值。

使用基本路径测试技术设计测试用例的步骤如下：

(1) 画出程序的控制流图。

(2) 计算流图的环路复杂度。

(3) 确定独立路径的基本集合。

(4) 设计测试用例。

8.4.3　条件测试

条件测试方法注重于测试程序中的条件，通过检测程序条件中的错误来发现程序中的其他错误。最简单的条件测试策略就是分支测试，它保证程序中每个简单条件，以及取“真”、取“假”的分支都至少被执行一次。

条件测试的目的不仅是检测程序条件中的错误，而且还要检测程序中的其他错误。如果程序 P 的测试集能有效地检测 P 中的条件错误，则它很可能也可以有效地检测 P 中的其他错误。此外，如果一个测试策略对检测条件错误是有效的，则很可能该策略对检测程序的其他错误也是有效的。

例如，程序中的关系表达式是 $X>Y$，那么，X、Y 的值可取 $X>Y$、$X=Y$、$X<Y$ 三种，查看表达式结果，一方面验证结果是否正确，另一方面可发现条件语句中的问题。

8.4.4　循环测试

循环测试专注于测试循环结构的有效性，根据循环的复杂性，可分为简单循环(见图 8-7)、嵌套循环(见图 8-8)和串接循环(见图 8-9)三种。

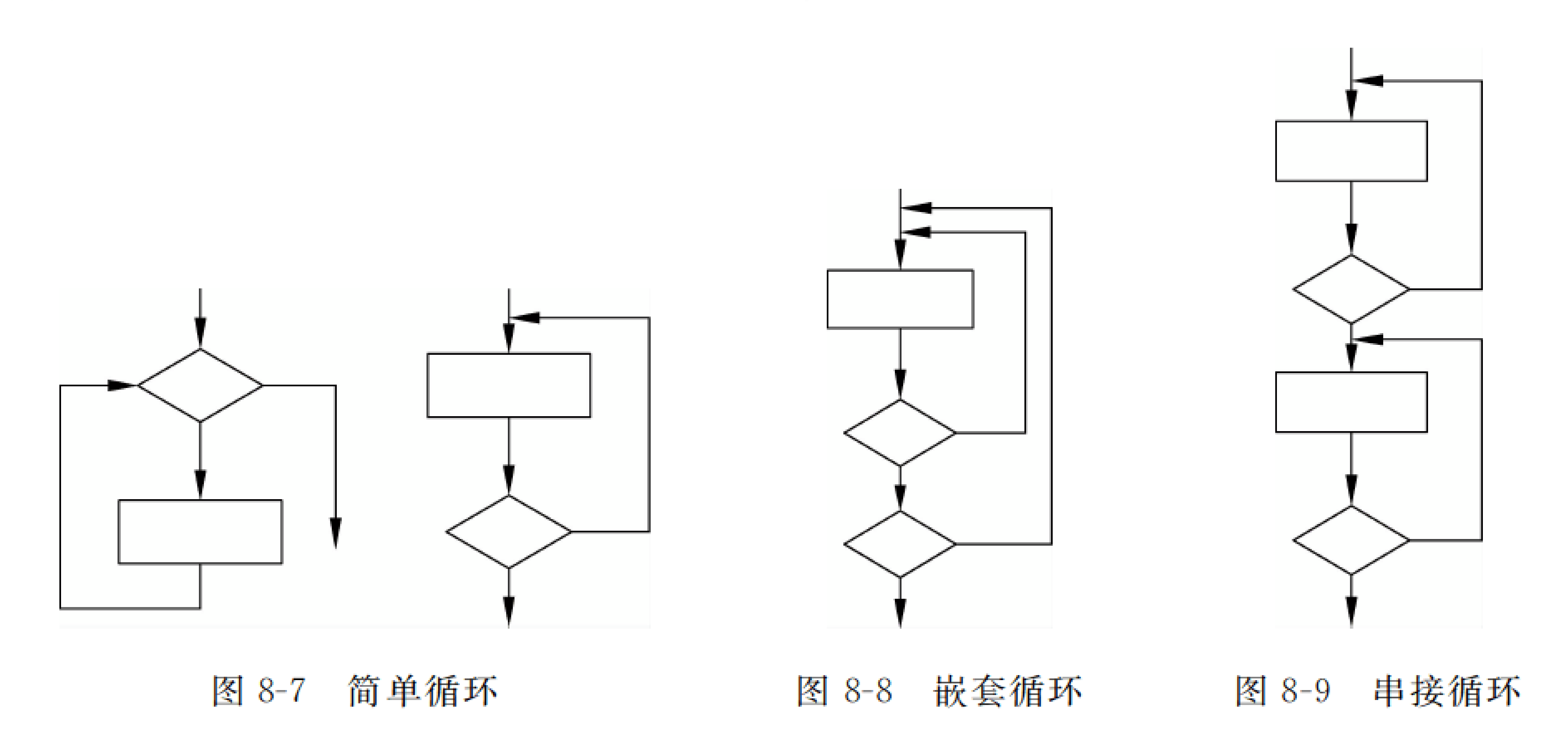


图 8-7　简单循环　　图 8-8　嵌套循环　　图 8-9　串接循环

下列测试集可以用于简单循环，其中 N 是允许通过循环的最大次数。

- 跳过整个循环。
- 只执行一次循环。
- 执行两次循环。
- 执行 M 次循环，其中 $M<N$。

- 执行 $N-1$、N、$N+1$ 次循环。

如果将简单循环的测试方法用于嵌套循环，随着嵌套层数的增加，测试数会呈几何级增长，导致测试数目过大，增加测试困难。B. Beizer 提出了一种能够减少测试数目的方法。

- 从最内层循环开始，将其他循环设置为最小值。
- 对最内层循环使用简单循环测试，而使外层循环的迭代参数最小，并增加其他的测试用例来测试范围外或排除的值。
- 由内向外构造下一个循环的测试，但其他的外层循环为最小值，并使其他的嵌套循环为"典型"值。
- 继续测试，直到测试完所有的循环。

如果串接循环的循环都彼此独立，可以使用简单循环测试策略来测试串接循环。但是，如果两个循环串接起来，而第一个循环的循环计数是第二个循环的初始值，则这两个循环并不是独立的。如果循环不独立，推荐使用嵌套循环的方法进行测试。

8.5 灰盒测试方法

灰盒测试，是介于白盒测试与黑盒测试之间的。可以这样理解：灰盒测试关注输出对于输入的正确性，同时也关注内部表现。但这种关注不像白盒那样详细、完整，只是通过一些表征性的现象、事件、标志来判断内部的运行状态，有时候输出是正确的，但内部其实已经错误了。这种情况非常多，如果每次都通过白盒测试来操作，效率会很低，因此需要采取这样的一种灰盒测试的方法。

灰盒测试通常与 Web 服务应用一起使用，因为尽管应用程序复杂多变，并不断发展进步，Internet 仍可以提供相对稳定的接口。由于不需要测试者接触源代码，因此灰盒测试不存在侵略性和偏见。

灰盒测试相对白盒测试更加难以发现并解决潜在问题。尤其在一个单一的应用中，通过白盒测试可完全掌握系统的内部细节。灰盒测试结合了白盒测试和黑盒测试的要素，它考虑了用户端、特定的系统知识和操作环境。它在系统组件的协同性环境中评价应用软件的设计。灰盒测试由方法和工具组成，这些方法和工具取材于应用程序的内部知识和与之交互的环境，能够用于黑盒测试以提高测试效率、错误发现和错误分析的效率。

8.6 测试过程与测试文档

在测试一个大规模的软件系统之前，测试人员应该清楚测试的整个过程。

8.6.1 测试过程

测试过程(Testing Procedure)指设置、执行给定测试用例并对测试结果进行评估的一系列详细步骤。测试过程可分为四个步骤：单元测试、集成测试、确认测试和系统测试。

(1) 单元测试集中对用源代码实现的每一个程序单元进行测试，检查各个程序模块是

否正确地实现了规定的功能。

(2) 集成测试把已测试过的模块组装起来，主要对与设计相关的软件体系结构的构造进行测试。

(3) 确认测试则是要检查已实现的软件是否满足了需求规格说明中确定了的各种需求，以及软件配置是否完全、正确。

(4) 系统测试把已经经过确认的软件纳入实际运行环境中，与其他系统成分组合在一起进行测试。

整个软件测试步骤如图 8-10 所示。

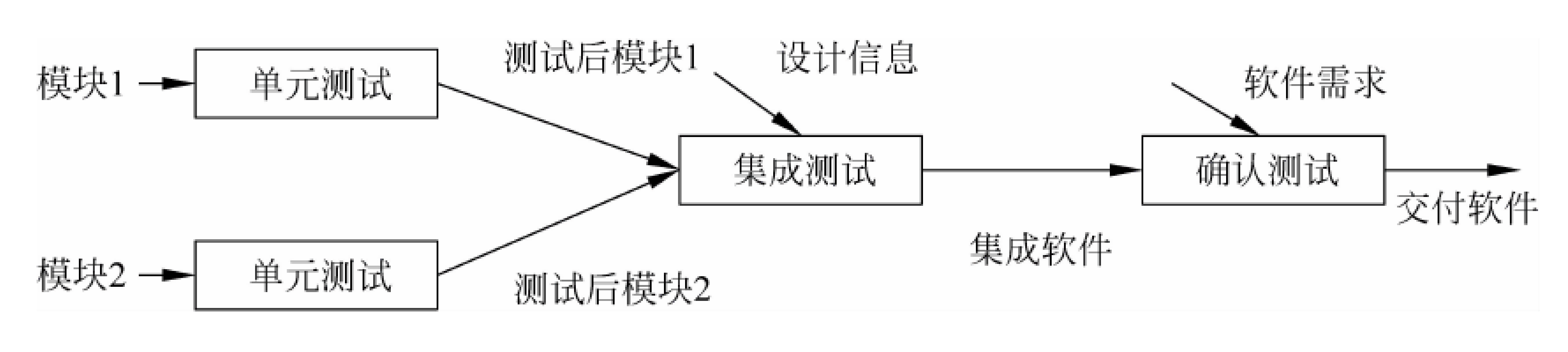


图 8-10　软件测试步骤

软件测试过程中需要三类信息：软件配置、测试配置和测试工具。软件配置包括需求规格说明、软件设计规格说明、源程序等。测试配置包括测试方案、测试用例、测试驱动程序等。测试工具指相关计算机辅助测试的工具，如测试数据自动生成工具、静态分析程序、动态分析程序、测试结果分析程序、驱动测试的测试数据库。整个测试过程如图 8-11 所示。

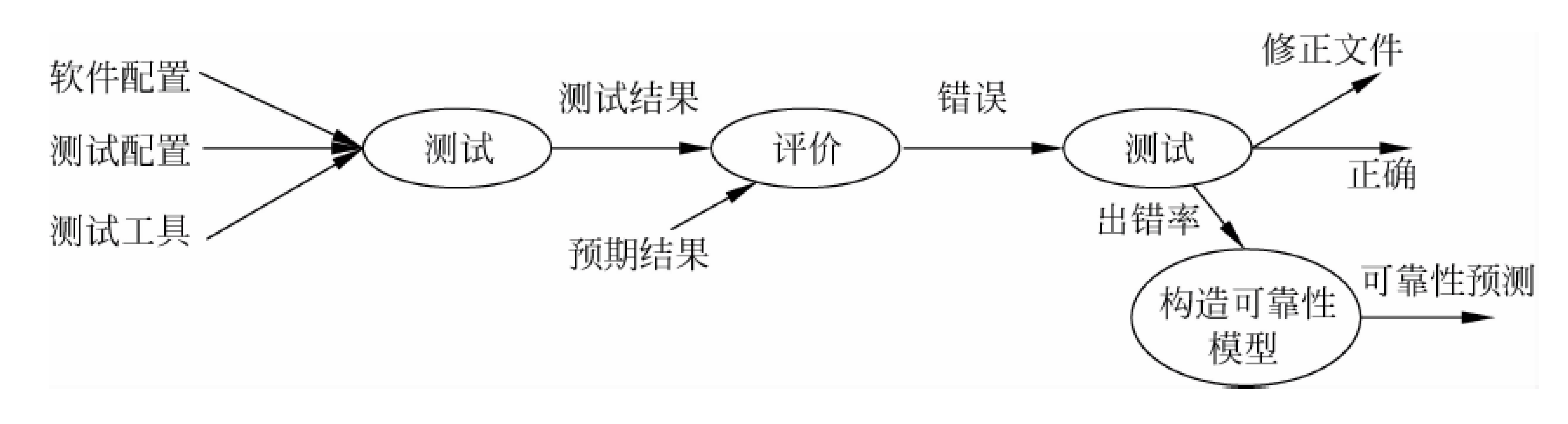


图 8-11　测试过程

8.6.2　测试文档

软件测试需要的文档有以下五个：

(1) 测试方案。主要设计测试什么内容和采用什么样的方法，经过分析，在这里可以得到相应的测试用例列表。

(2) 测试执行策略。主要包括哪些可以先进行测试，哪些可以放在一起进行测试之类的策略。

(3) 测试用例。主要根据测试用例列表，写出每一个用例的操作步骤、紧急程度。

(4) bug 描述报告。主要包括对测试环境的介绍、预置条件、测试人员、问题出现的操作步骤和当时测试的现场信息。

(5) 整个项目的测试报告。从设计和执行的角度，对项目测试情况进行介绍，从分析中总结此次设计和执行做得好的地方和需要努力的地方，并对此项目进行一个质量评价。

8.7 本章小结

软件测试是保证软件可靠性的主要手段,它的根本任务是发现并排除在分析、设计和编程过程中所产生的各种错误。软件测试可分为白盒测试、黑盒测试和灰盒测试。白盒测试技术主要包括逻辑覆盖、基本路径测试、条件测试和循环测试。黑盒测试技术主要包括等价类划分、边界值分析、错误推测法等。

软件测试的步骤包括单元测试、集成测试、确认测试和系统测试。单元测试着重鉴定一个模块的功能;集成测试要把多个模块组成为完整的软件;确认测试是根据需求规格说明书中定义的全部功能和性能要求,确认软件是否达到了要求;系统测试则要证明软件在并入更大的系统时能有效地工作。

习题 8

1. 什么是软件测试?软件测试的基本准则是什么?
2. 软件测试的目的是什么?为什么把软件测试的目的定义为只是发现错误?
3. 简述黑盒测试方法。
4. 简述白盒测试方法。
5. 简述测试的过程。
6. 单元测试、集成测试、确认测试各自的主要目标是什么?它们之间有什么不同?相互之间有什么关系?
7. 根据你自己的经验,总结在程序调试中常用的纠正差错的方法。

第9章 软件实施与维护

9.1 软件产品的分类

软件产品总体可分为系统软件、支持软件和应用软件三大类。

系统软件(System Software)是指控制和协调计算机及外部设备,支持应用软件开发和运行的系统,是不需要用户干预的各种程序的集合,主要功能是调度、监控和维护计算机系统;负责管理计算机系统中各种独立的硬件,使得它们可以协同工作。系统软件使得计算机使用者和其他软件将计算机当作一个整体而不需要顾及到底层每个硬件是如何工作的。

一般来讲,系统软件包括操作系统和一系列基本的工具,是支持计算机系统正常运行并实现用户操作的那部分软件,一般是在计算机系统购买时随机携带的,也可以根据需要另行安装。

系统软件的主要特征是:

- 与硬件有很强的交互性;
- 能对资源共享进行调度管理;
- 能解决并发操作处理中存在的协调问题;
- 其中的数据结构复杂,外部接口多样化,便于用户反复使用。

支持软件(Support Software)又被称为软件开发环境,是支撑各种软件开发与维护的软件,是一组软件工具的集合,主要包括环境数据库、各种接口软件和工具组。著名的软件开发环境有 IBM 公司的 Web Sphere,微软公司的 Studio. NET 等。工具组主要包括一系列基本的工具(比如编译器、数据库管理、存储器格式化、文件系统管理、用户身份验证、驱动管理、网络连接等方面的工具)。

应用软件(Application Software)是为了某种特定的用途而被开发的软件,是用户可以使用的各种程序设计语言,以及用各种程序设计语言编制的应用程序的集合,分为应用软件包和用户程序。应用软件包是利用计算机解决某类问题而设计的程序的集合,供多用户使用。应用软件是为满足用户不同领域、不同问题的应用需求而提供的那部分软件。它可以是一个特定的程序,比如一个图像浏览器;也可以是一组功能联系紧密、可以互相协作的程序的集合,比如微软的 Office 软件;也可以是一个由众多独立程序组成的庞大的软件系统,比如数据库管理系统。总之,应用软件可以拓宽计算机系统的应用领域,扩展硬件的功能。

9.2 软件产品的发布

9.2.1 产品发布策略

产品的发布时机是由市场利润、开发进度、产品功能与质量、版本管理状态、客户可接受程度等多方面的因素决定的。

微软“基于版本发布”的指导原则中的第一项内容就是 Trade-of Decision，即“折中决定”。该决定的指导思想是：当产品的“可靠性”介于“最优”与“客户可以接受”两者之间时，就可以发布了。

微软“基于版本发布”的指导原则中的第二项内容，就是项目管理团队、开发团队和测试团队三方都签字确认终结产品的开发，冻结该产品的版本(终结与冻结工作由配置管理员执行)，该产品才能被发布。

9.2.2 产品发布流程规范

严格按照软件产品发布流程发布软件版本是建立和完善软件产品版本控制、保证软件产品质量、确保应用软件正常发布的关键。

1. 发布准备

当软件产品的 Beta 版本测试合格，并且项目管理团队、开发团队和测试团队三方都签字确认终结该产品的开发后，开发人员首先要确定发布的准备工作和发布的日期，这部分准备工作应包含以下内容：

- 原有 bug 是否已被彻底排除；
- 新增模块在功能上是否已达到设计要求；
- 修改了什么，增加了什么；
- 所做的改变带来的影响。

然后，企业的高层管理人员应向市场与销售中心下达《产品发布通知单》，市场与销售中心须做如下准备：

- 编写培训教材；
- 产品包装设计；
- 产品母盘制作；
- 产品光盘刻录；
- 软件资料印刷；
- 销售人员培训；
- 发布产品检验；
- 发布产品交付；
- 确定发布方式。

2. 撰写文档

开发人员确定所发布内容中是否有新增功能。若有，则需要撰写一份需求文档，交给测试人员，否则修改 bug 状态，并发送测试通知单，告知测试人员。需求文档的内容如下：

- 所做的改动有哪些；
- 修改原有 bug 或新增模块的设计目标。

3. 全面测试

测试人员在收到测试通知单或需求文档后，应进行全面、完善地测试。如果通过测试，则应将测试报告发送给开发经理，并修改 bug 状态。否则，将测试结果反馈给开发人员，测试结果中应包含以下内容：

- 原有 bug 的排除情况或新增模块的 bug 情况；
- 发现 bug 的测试用例。

4. 发布确认

所有程序由测试人员进行确认测试后，若发现检查系统内登记的所有 bug 都已经被修正，或者遗留的 bug 不影响系统的使用，则由系统维护员审核 bug 解决情况或新增模块使用情况，如果符合要求，则准许发布，如果有严重 bug 未被排除（级别为必须修正）则不能发布。

5. 质量认定

测试负责人编写发布版本产品质量报告以进行质量分析和总结。

6. 源码、文档入库

源码包括 JavaScript 脚本、数据库创建脚本（含静态数据）、编译构建脚本和所有源代码。文档包括需求、设计、测试文档、安装手册、使用手册、二次开发手册、产品介绍（PPT 文件）等。

7. 程序打包

（1）Build Master 进行程序打包，给新版本的源代码标记文档版本 tag，方便代码回滚。比如，发布版本为 1.0.0，则给该软件源代码也打一个与发布版本相同名字的 tag——1.0.0。这样做的一个好处是，在目前的软件的基础上做了修改并发布新的版本后，如果需要检出某个版本的源代码，则可以通过这个版本的 tag 来进行，代码的修改可以在该版本上进行。

（2）Build Master 对新发布的软件源代码进行锁定，不允许开发人员在软件发布之后提交源代码，直到有新版本需求修改再给开发人员开放提交权限。这样做的好处是避免开发人员随意修改和提交源代码，确保源代码服务器上的源代码版本与当前最新的发布版本一致。

8. 发布产品

新建产品发布计划，填写配置项，执行发布计划(发布产品)。

9. 编写发布说明

编写发布说明 readme.txt(或者 release note)。readme.txt 的内容应该包括产品版本说明；产品概要介绍；本次发布包含的文件包和文档说明；本次发布包含或者新增的功能特性说明；遗留问题和影响说明；版权声明和其他需要说明的事项。

10. 正式发布通知

通知开发、测试、市场、销售各相关部门并附上产品发布说明和产品介绍。

11. 后续工作

产品发布后，在使用过程中可能还会发现一些 bug。在不影响正常使用的情况下，这些 bug 将在下一版本发布时解决；如果 bug 严重影响使用，必须打补丁或者按照流程重新发布。

12. 临时发布

软件产品未正式发布前，可能需要一个临时版本供开发人员或者用户应急使用，这时候需要临时发布一个版本。这个版本只包括基本的程序包和必要的使用说明。临时发布需要通知相关开发和测试人员；Build Master 需要为源码和文档打 tag 标记。

9.2.3 产品发布方式

软件企业市场与销售中心要通过各种媒体进行产品发布，以扩大影响、吸引客户、占领市场。不管是哪一类软件产品，其产品发布的方式不外乎下面三种：

(1) 聘请有关领导、新闻媒体记者和各大客户代表，召开新闻发布会，宣布新产品的优点，描述其市场前景，现场演示，厂商给嘉宾和客人赠送产品资料。

(2) 在报纸、刊物、电视台、电台上做广告，宣传软件产品。

(3) 在各种交易会、展览会、博览会上租用摊位，展示软件产品。

当大型 IT 企业快要发布产品的时候，与该产品有关的工程师、程序员和测试人员都要随时待令，打开手机，随叫随到，解决产品中的任何问题。

9.3 软件产品的实施

软件产品的实施是一个软件产品从内部开发完成、产品发布，到系统正式运行之间的一个阶段过程。对于定制的行业应用软件，实施是产品开发的延续；对于大型复杂软件系统，实施是包括咨询服务在内的软件产品的组成部分。

对于被称为“盒装软件”的商业软件，其实施过程现在已经非常简单。如 Windows 8 或

Microsoft Word 2010 的系统安装，已经做得非常友好。而对于一些类似财务软件这样的商业软件，实施过程就不那么容易了，因为在安装时需要设置一些系统参数、权限管理、基本数据维护等。还有一些更复杂的大型应用软件系统，如企业资源计划系统（Enterprise Resource Planning，ERP），它的实施过程就是一个咨询、培训和方案实施的过程，远不是安装系统、设置参数的操作概念。

9.3.1 软件产品实施步骤

一个标准、完整的软件产品实施过程可分为以下六个步骤。

1. 实施准备

实施准备阶段将完成软件实施计划的制定，为后期的实施过程做好详尽的准备工作。该阶段的工作内容包括以下六个：

（1）成立相应的软件实施组织机构，确定各机构的相关人员。软件实施组织机构如表 9-1 所示。

表 9-1 软件实施组织机构

软件实施领导小组	组长：由用户方的领导担任
	副组长：由软件实施方的领导担任
	成员：由用户方的相关部门领导、用户方或软件实施方分别指定的项目经理及实施顾问组成
软件实施小组	组长：由软件实施方指定的项目经理担任
	成员：由软件实施方指定的实施人员组成
软件业务小组	组长：由用户方指定的项目经理担任
	成员：由用户方指定的业务骨干及实施配合人员组成

（2）对软件实施领导小组、软件业务小组的用户方成员组织一次系统级的培训工作（只进行系统讲解，不进行上机操作），向其传达待实施系统的理念、总体架构、系统特点等。

（3）确定本次软件实施的策略（包括软件实施的整体目标、实施范围、实施策略等）。

（4）确定双方的工作范围及职责划分。

（5）编制软件的实施计划。

（6）制定软件实施过程中应遵循的规范、标准等。

2. 业务交流

根据实施准备阶段确定的实施策略，按照实施计划的要求完成下列工作：

（1）完成对待实施用户原有系统的调研，掌握原系统的功能体系结构及数据结构，并确定新老系统的数据接口，编制新老系统的接口方案。

（2）通过讲解、沟通的方式，完成与待实施用户的业务交流，确定重组方案。

（3）了解软硬件及网络配置情况，结合待实施软件产品的软硬件及网络配置要求，完成待实施用户的软硬件及网络配置规划。

（4）了解待实施用户的组织架构及权限控制要求，编制系统的授权方案。

(5) 整理并编制系统的数据准备方案，安排用户方进行系统的基础数据准备。

(6) 结合上述工作，编制软件实施规划建议书。

(7) 制定软件的实施培训计划。

(8) 完成软件培训环境及培训教材的准备工作。

业务交流阶段的成果为软硬件及网络配置规划；软件实施规划建议书；新老软件系统数据接口方案；系统数据准备方案；软件实施培训计划和软件实施培训教材。

3. 软件实施培训

在软件实施培训阶段，首先完成软件培训环境的搭建，包括培训网络的搭建、培训系统的安装及初始化、培训用户及权限的初始化等；然后，采用讲解与上机练习相结合的方式，分别完成不同级别的实施培训工作；最后收集、汇总、整理本次培训，完成软件实施培训记录。

具体的实施培训包括以下三个：

(1) 系统管理员培训。针对系统管理员进行有关基础数据维护、工作流程设置、标准模板定义、授权定义及分配等系统级的应用操作培训。

(2) 数据库管理员培训。针对数据库管理员进行有关数据库的安装和维护，系统应用数据的备份、恢复等方面的知识培训。

(3) 操作员培训。针对软件系统的各个功能模块所涉及的操作员组织进行培训，使其了解该功能模块的业务处理规范及应用操作方式。

4. 系统初始化

系统初始化阶段将获得系统安装、升级记录清单及新老系统切换计划，其具体工作内容为：系统运行环境的搭建(包括网络平台的构建、软硬件的配置、数据库的安装等)；系统的安装、初始化(包括软件模块的安装和配置，数据库的定义及系统基础数据的初始化等)；应用基础数据的整理及导入；业务流程的设置；用户角色的定义及系统权限的分配；新老系统的接口设计与实现；新老系统切换方式(见图 9-1)的确定和编制。

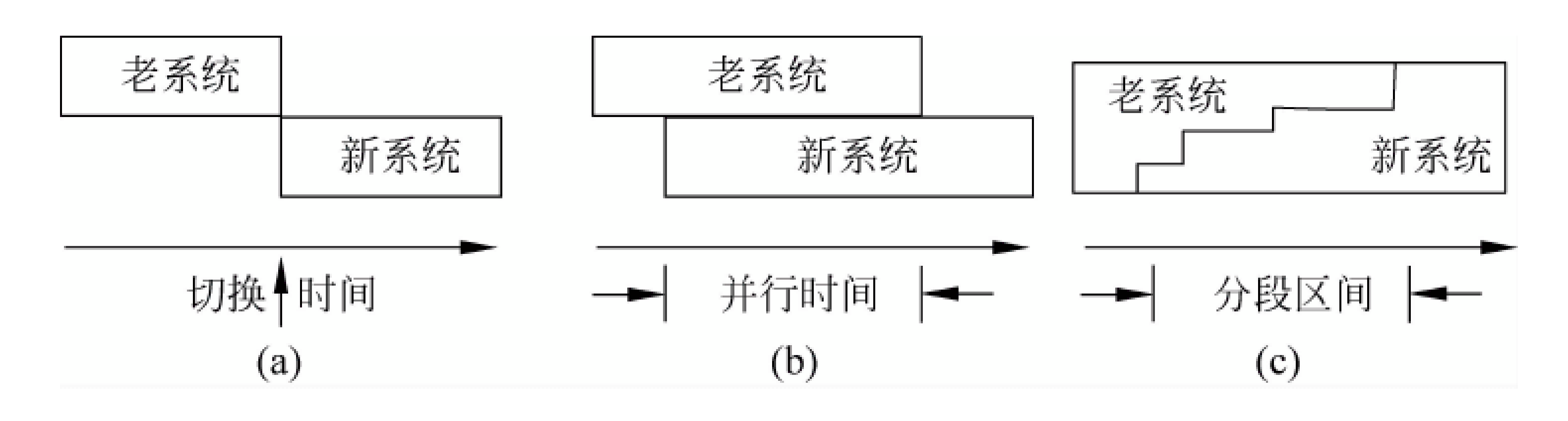


图 9-1 新老系统切换方式

5. 新老系统切换及试运行

- 按照新老系统切换计划，完成新老系统的切换，将新系统投入试运行。
- 协助用户完成系统试运行过程中的日常维护工作，并提供技术支持。
- 收集、整理系统试运行过程中的用户反馈意见。
- 根据系统的试运行情况，协助用户编制系统试运行报告。

- 在达到试运行目标后，向用户提出系统验收申请，并完成系统验收前的准备工作。

该阶段在得到用户提供的系统试运行报告后，可进一步制定系统验收计划。

6. 系统验收

系统验收包括组织、完成新系统的验收工作，协助甲方用户编制系统验收报告；系统验收后转入实际运行，提供系统运行过程中的日常维护与技术支持工作。最后，由用户给出系统试运行报告。

在具体的实施过程中，因软件规模或性质的不同，其实施过程也存在一定差异，可根据实际情况对上述步骤进行剪裁。

9.3.2 实施过程中的整改处理

在软件实施过程中可能遇到系统整改（包括需求的新增、变更，系统的完善等）的情况，则必须严格按照下列的处理步骤进行（见图 9-2）。

（1）在实施过程中发现系统问题时，用户需填写《系统问题报告单》并定期汇总到用户方项目经理。

（2）双方项目经理定期（如每周一或周五）对收集的《系统问题报告单》进行汇总、分析，提出系统本次整改的内容及方法，编制《系统整改方案（初稿）》，并将其上报软件实施领导小

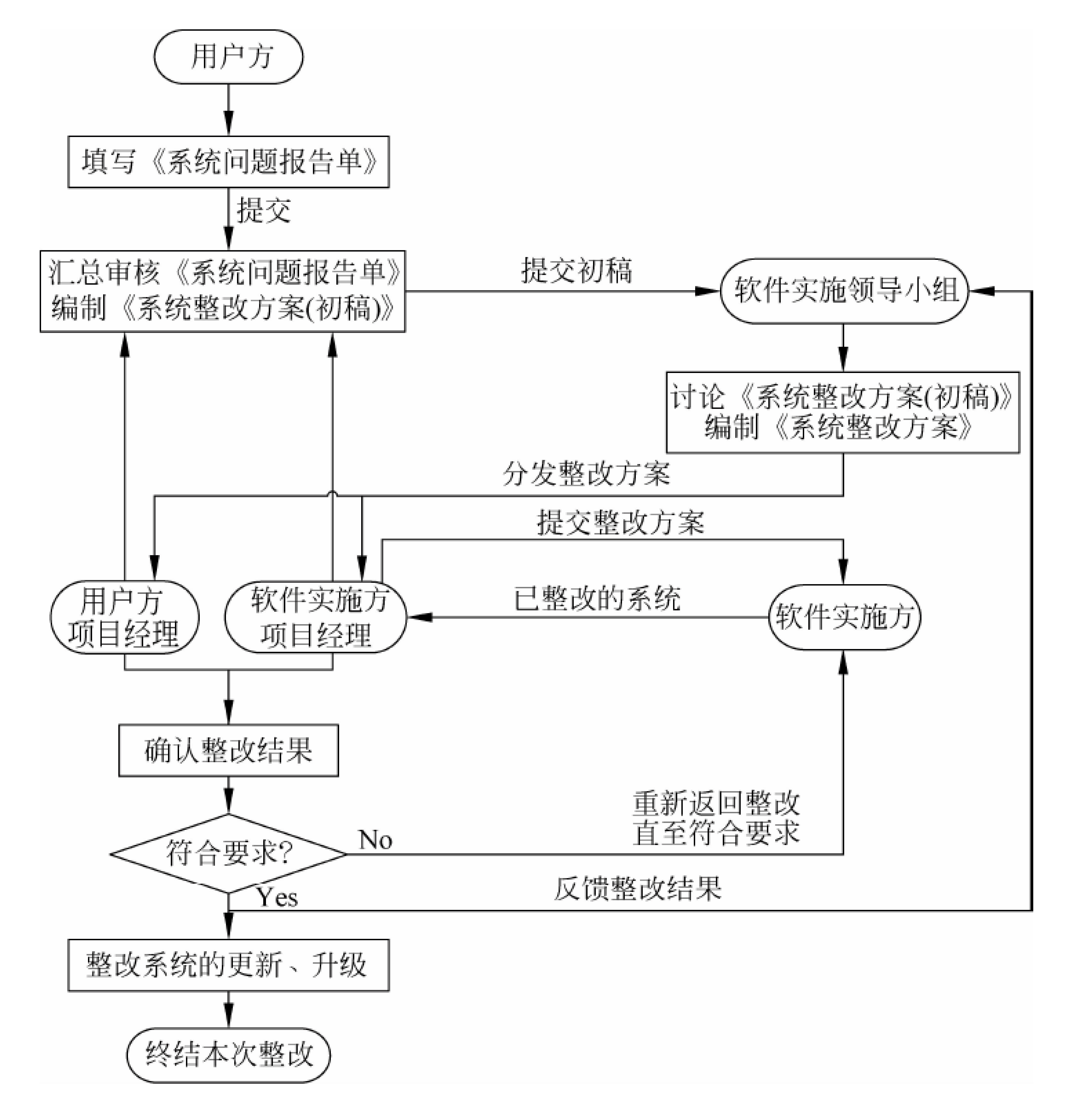


图 9-2　整改处理步骤

组审批。

(3) 软件实施领导小组在收到项目经理上报的《系统整改方案(初稿)》后，开会逐条讨论，确定最终的《系统整改方案》，并将其下发给双方的项目经理。

(4) 软件实施方项目经理在收到软件实施领导小组确定的最终《系统整改方案》后，及时传递给公司；由公司安排人员对系统进行整改，并由用户方项目经理督促其整改进度。

(5) 系统整改完毕并确认无误后，由软件实施方项目经理组织用户方项目经理及相关人员对整改结果(根据最终的《系统整改方案》)逐条进行确认，若有不符合要求的则需要返回步骤(4)重新进行，直至全部符合要求；将整改结果反馈给软件实施领导小组。

(6) 由软件实施方项目经理组织实施人员进行已整改系统的安装，完成系统的版本更新。

9.4 软件维护的传统方法

软件产品完成内部开发并交付给用户使用之后，就进入了运行维护阶段。

随着用户需求和软件运行环境的变化，以及软件自身潜在问题的逐渐显露，对其进行维护便成了不可或缺的环节。软件维护是软件生命周期的最后一个阶段，是对原有系统的一种修改，其基本任务是保证软件在一个相当长的时期能够正常运行。软件维护所需要的工作量很大，内容复杂而且艰巨，据统计和估测结果表明，很多软件后期维护成本占到了整个生命周期成本的40%～70%，有时竟高达软件总费用的80%。目前国外许多软件开发组织把60%以上的人力用于维护已有的软件，而且随着软件数量的增多和使用寿命的延长，这个百分比还在持续上升。

由于软件在使用过程中，需要不断地发现和排除错误，以及适应新的要求，所以软件工程的主要目的就是提高软件的可维护性，减少软件维护所需要的工作量，降低维护成本。

9.4.1 软件维护的定义

软件交付使用后的修改等工作被称为软件维护。确切地说，软件维护就是在软件已经交付使用之后，为了改正软件中的错误或增加功能以适应新需求而修改软件的过程，一般不包括软件体系结构上的重大改变。根据要求进行维护的原因的不同，软件维护可以分为以下四类。

1. 改正性维护

改正性维护是一种限制在原需求说明书的范围之内，修补软件缺陷的维护。因为软件是在有限的时间和经费限制前提下进行测试，由于受到测试技术和手段的限制，不论经过多少次严格测试，可能仍然不彻底、不完全，必然会有部分隐藏的错误遗留到运行阶段，这些隐藏下来的错误在某些特定的使用环境下就会暴露出来，甚至会导致软件系统发生故障。为了识别和纠正软件错误、改正软件性能上的缺陷、排除实施中的误操作，应当进行的诊断和改正错误的过程就叫做改正性维护。

2. 适应性维护

随着计算机的不断发展，软件在使用过程中，其外部环境(新的硬、软件配置)、数据环境(数据库、数据格式、数据输入输出方式、数据存储介质)等可能发生变化。适应性维护，就是为了使软件适应操作环境变化而进行的修改软件的活动，是既必要进行的又是经常要进行的维护活动。

3. 完善性维护

在软件的使用过程中，用户往往会提出增加新功能或修改已有功能的建议，还可能提出一般性的改进意见。为了满足这些要求，需要修改或再开发软件，以达到适应新要求、扩充软件功能、增强软件性能、改进软件效率、提高软件的可维护性等目的。这种情况下进行的维护活动叫做完善性维护，是为了改善、加强系统的功能和性能，以满足用户新的要求而对软件进行的维护活动。

4. 预防性维护

预防性维护即软件再工程，需要根据现有的信息对未来的环境变化进行预测，再根据预测的结果采取相应的解决措施，是为了提高软件未来的可维护性、可靠性等质量目标，为以后进一步改进软件奠定良好基础。采用先进的软件工程方法对需要维护的软件或软件中的某一部分(重新)进行设计、编制和测试，这就是预防性维护。

预防性维护具有一定的风险，因为它是根据目前软件运行的内外环境等信息提出的、能够提高软件的可维护性和可靠性的一些措施，具有很大的局限性。所以，预防性维护在所有维护工作中所占比例很小。

可以将预防性维护看成是产品开发的延续，在这个意义上说，改正性维护、适应性维护、完善性维护是被动的，而只有预防性维护是主动的。

从上述关于软件维护的定义可以看出，软件维护绝不仅限于纠正使用中发现的错误。事实上在全部维护活动中，一半以上是完善性维护，即大部分维护工作是改变和加强软件而不是纠错。国外的统计数字表明，完善性维护占全部维护活动的 50%～66%，改正性维护占 17%～21%，适应性维护占 18%～25%，其他维护活动只占 4%左右，如图 9-3 所示。

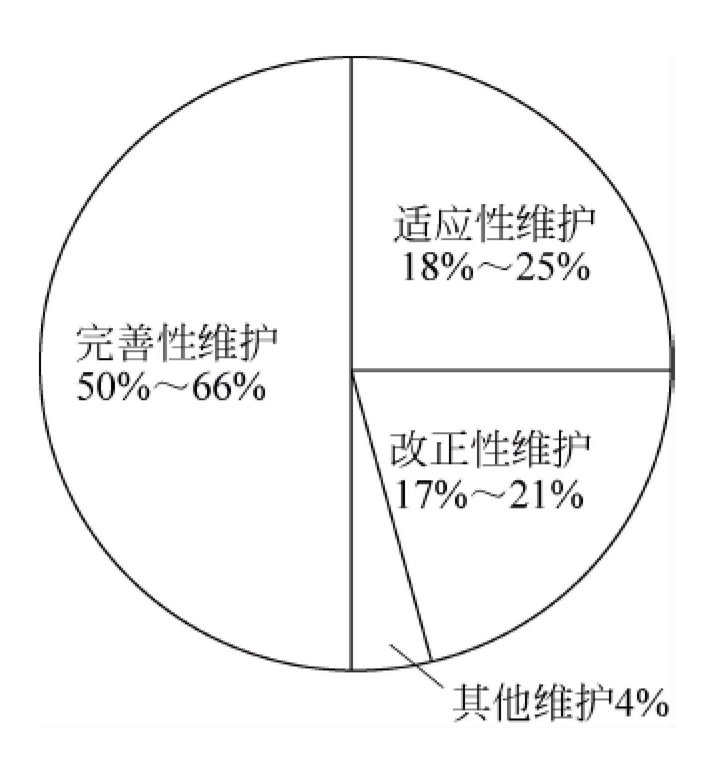


图 9-3 软件维护比重

应该注意，上述四类维护活动都必须应用于整个软件配置，维护软件文档和维护软件的可执行代码是同样重要的。

9.4.2 软件维护的特点

1. 工作量大

软件维护是软件生产性活动中延续时间最长、工作量最大的活动之一。大、中型软件产

品的开发期一般为1～3年,运行期可达5～10年,而软件维护实际是一个修改和简化的软件开发过程。软件开发的所有环节,如分析、设计、实现、测试等几乎都要在维护活动中用到。因此,在整个软件的有效运行期内对软件产品进行维护需要很大的工作量。

通常,把软件维护的工作量分为生产性活动和非生产性活动。生产性活动是用于分析评估、修改设计、修改代码等工作所需的工作量;非生产性活动则是用于理解程序代码的功能,解释数据结构、接口特点、性能限度等工作所需的工作量。

维护工作量的估算模型如下:

$$M = P + K\mathrm{e}^{(c-d)}$$

其中,M是维护所用总工作量;P是生产性工作量;K是经验常数;c是维护复杂度,由软件本身的复杂度、软件的设计质量、文档化的程序等因素决定;d是维护人员对软件的熟悉程度。

模型中的第一项P是生产性工作量,第二项$K\mathrm{e}^{(c-d)}$是非生产性工作量。估算模型表明,维护工作量与软件的维护复杂度成指数关系,如果软件开发没有运用软件工程方法学,而且原来的开发人员不能够参与到维护工作之中,则维护工作量将呈指数性增加。

2. 代价高昂

在过去的几十年中,软件维护的费用逐年上升。20世纪70年代,用于维护已有软件的费用约占软件总预算的35%～40%;20世纪80年代,上升为40%～60%;近年来,该部分已上升到70%～80%左右。维护费用只是软件维护最明显的有形代价,还有其他不易估量的无形代价更应该引起注意。

- 可用的资源被软件维护所占用,以至耽误甚至丧失了开发的良机。
- 未能及时满足用户的维护要求时,会引起用户不满。
- 在维护时改动软件,引入了潜在故障,降低了软件质量。
- 抽调软件工程师从事维护工作,对新的开发过程造成混乱。
- 导致生产率的大幅下降(这种情况在维护旧程序时常常遇到)。

3. 问题很多

维护阶段所遇到的绝大多数问题,都与软件设计、开发、测试阶段所采用的方法、技术等有直接关系,同时与维护工作的性质也有一定的关系。软件开发是采用急功近利的态度,还是放眼未来的态度,对软件维护工作的影响极大。一般说来,软件开发若不采用软件工程思想并严格遵循软件开发标准,软件维护就会遇到许多困难。

下面列出了软件维护过程中常见的典型问题。

1) 文档不全或前后不一致

主要表现在需要维护的软件没有合格的对应文档,文档前后不一致,或文档与程序之间不一致。软件开发过程中经常会出现修改程序而忘记修改相关文档,或者修改了某一个文档,却没有修改与之相关的其他文档等现象,文档的不一致性往往就是由于这种对文档管理不严格的操作造成的。解决文档不一致性的方法就是要完善文档管理工作,使文档容易理解并且和程序代码完全一致。

2）理解源程序非常困难

如果源代码编写过程中没有严格遵循合适的开发规范，或者注释不全，则会使得源代码非常难以被理解。并且，开发人员与维护人员大多数不是同一个人，一般开发人员都有这种体会，理解别人写的程序往往非常困难，修改别人的程序还不如自己重新编写程序，而且困难程度随着软件配置成分的减少而会迅速增加。

3）无法获得开发人员的帮助

通常，从事开发和维护的工作人员是不同的，开发人员也许已从事其他新软件的开发工作，维护人员通常无法获得开发人员的帮助，这就意味着在维护阶段不可能有开发人员来仔细说明软件，从而造成软件整体理解和追踪完整创建过程的困难。又因为维护阶段持续时间较长，开发时所使用的工具、方法和技术可能与当前有较大差异，这些都会对维护工作造成困难。

4）软件原有的设计缺陷

很多软件在开发时并未考虑将来的可维护性，在分析和设计阶段没有使用模块独立原理的设计方法学，导致软件存在可维护性缺陷，既难于理解又容易发生错误，对软件的维护带来严重问题。

5）对软件维护的误解

软件维护不是一项吸引人的工作，成功的维护也只是保证他人开发的系统能正常运行，形成这种对维护工作的误解很大程度上是因为维护别人开发的软件经常遭受挫折，使得维护人员缺乏成就感。

9.4.3 软件维护的过程

维护过程实际上是简化和修改了的软件开发过程，并且软件维护有关的工作通常很早就开始进行了。首先必须建立一个维护组织，然后确定维护报告内容，记录维护流程、保存记录，最后确定评估及复审标准。此外，还应当为每个维护申请确定一个标准化的事件序列。

1. 维护组织

鉴于软件维护自身的工作量大、持续时间长、代价高昂和面临问题很多的特点，要想达到较好的维护效果，就应当建立专门的维护组织（见图 9-4），在维护活动开始之前明确维护责任，减少维护过程中可能出现的混乱。对于一些小的软件开发团体而言，虽然通常并不需要建立正式的维护机构，但非正式地委托责任也是绝对必要的。

一个标准的维护组织一般由以下人员组成：维护管理员、系统管理员、维护决策机构、配置管理员和维护人员。

用户的每个维护申请都必须提交给协调维护活动的维护管理员，由维护管理员转交给熟悉该产品的系统管理员去评估。系统管理员一般都是对程序（某一部分）特别熟悉的技术人员，他们对维护申请及可能引起的软件修改进行评估，并向维护决策机构（一个或一组管理者）报告，最终由该机构决定是否应该进行维护活动。

软件开发单位根据自身规模的大小，可以指定一名高级管理人员担任维护管理员，或者建立由高级管理人员和专业人员组成的维护决策机构，管理本单位开发的软件维护工作。

管理的内容应包括对申请的审查与批准、维护活动的计划与安排、人力资源的分配、批准并向用户分发维护的结果，以及对维护工作进行评价与分析。

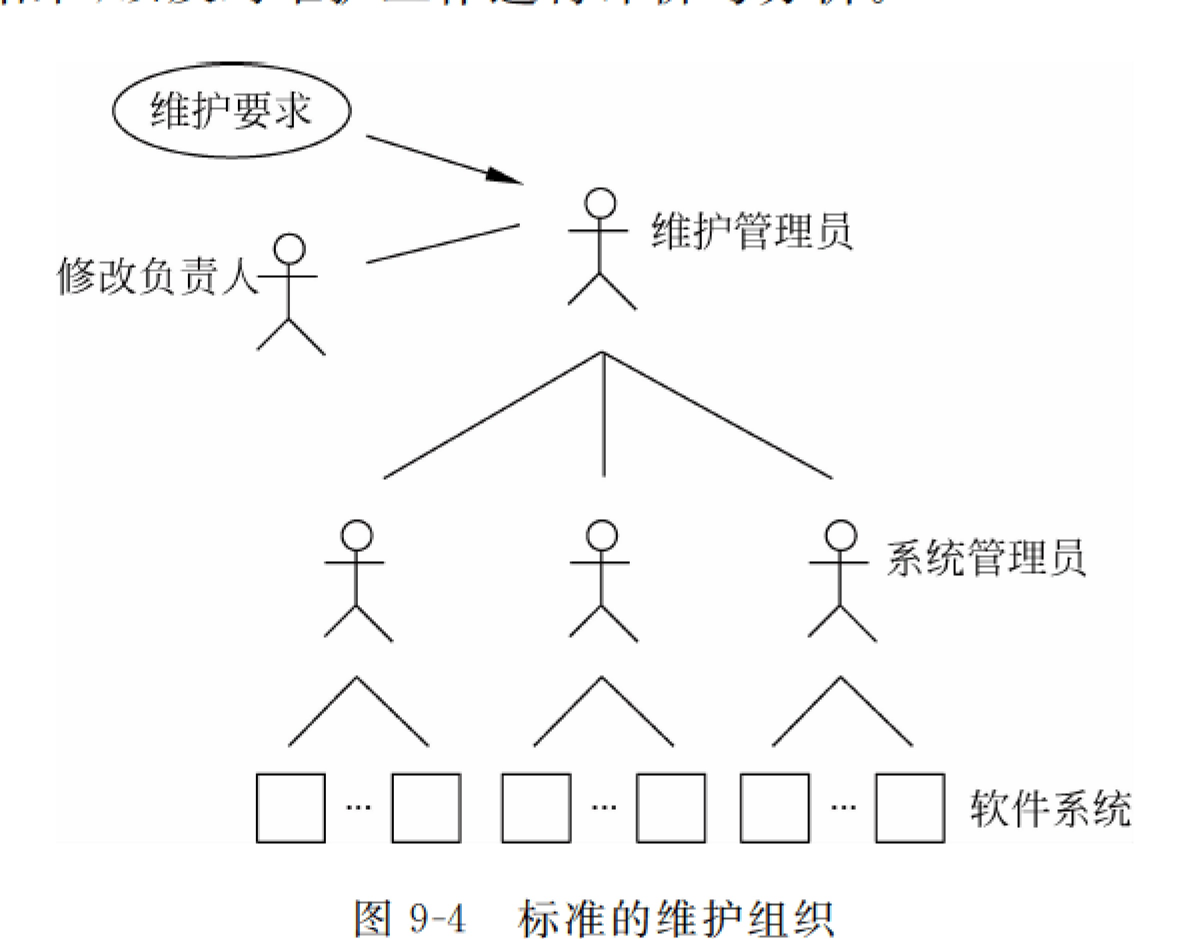


图 9-4 标准的维护组织

2. 维护报告

当用户有维护要求时，应该使用标准的书面格式去提出维护申请。维护管理员通常会给用户提供空白的维护申请报告表(有时被称为软件问题报告表)，由用户填写需要维护的相关内容。如果是改正性维护要求，用户必须完整地描述产生错误的情况，包括输入数据、全部输出数据、错误清单以及其他有关信息。对于适应性或完善性的维护要求，则必须由用户提出一个简短的需求说明书，列出所有希望改进的内容。用户提交的维护申请报告表将由维护管理员接收，并转交给系统管理员进行评估。

维护申请报告是从软件组织外部提交来的文档，是计划维护工作的基础。软件组织内部应针对维护申请报告，相应地制定出一个软件修改报告，并提交给维护决策机构审查批准。该报告包含下述信息：

- 此次维护的性质；
- 该维护申请的优先级；
- 完成维护申请报告表中所提出的、需要的工作量；
- 预计修改后的状况。

3. 维护流程

不论是何种类型的软件维护，都需要进行同样的技术工作：

- 修改软件设计；
- 复查；
- 必要的源代码修改；
- 单元测试；
- 集成测试(包括使用以前的测试方案进行回归测试)；
- 验收测试；
- 复审。

不同类型的维护强调的重点不同,但是基本途径是相同的,维护事件流中的最后一个事件是复审,它将再次检验软件配置的所有成分的有效性,并且保证事实上满足了维护要求表中的要求。

在进行维护工作之前,首先应该确定要求进行的维护的类型。在确定维护类型时,维护人员应与用户反复协商,弄清申请维护报告中所述的内容,以及用户希望有何改进,再确定所需要进行的工作属于何种维护。因为在很多情况下,用户往往把所要求的维护看做是为了改正软件的错误(改正性维护),而开发人员可能把同一项要求看做是适应性或完善性维护。

对于改正性维护,应当从评估错误的严重程度开始。若是系统的某个关键功能无法正常运行,就属于严重软件错误,遇到此类情况,则应在系统管理员的指导下分派维护人员,立即开始分析问题。如果是错误不严重的改正性错误,其处理操作将和其他要求软件开发资源的任务一起统筹安排。

适应性维护和完善性维护申请采取相同的方法。但由于对商业策略、当今和今后软件产品方向等方面的考虑,不是所有完善性维护都会被接受。所有被接受的维护,就像安排一个开发任务一样(从所有意图和目标来看,它都属于开发工作),首先确定其优先次序,然后在维护活动中安排其位置、所需要的工作及时间。如果一项维护要求的优先级非常高,则应当立即开始维护工作。

如果出现了恶性的软件问题,即所谓的"救火"维护要求,则必须临时放弃正常的维护工作程序,立即把资源用来解决问题。这时既不对修改可能带来的副作用做出评价,也不对文档做出相应的更新,而是立即进行代码的修改。但是这种救火式的改正性维护,只在非常危急的情况下才采用,在全部维护中一般只占很小的比例。并且也不是取消,而是推迟了维护所需要的控制和评价。一旦危机消除,这些控制和评价活动必须继续进行,以确保当前的修改不会增加更为严重的问题。如果对一个组织来说,"救火"是常见的过程,那么必须怀疑它的管理能力和技术能力。

软件维护任务完成以后,一般还要对维护工作进行处境复查。通常,这样的复查将回答如下问题:

- 在当前状况下,设计、编码或测试的哪些方面还可以采用其他方法加以改进?
- 还没有应用哪些可用的维护资源?
- 此次维护工作的主要或次要障碍是什么?
- 维护申请报告中是否提出了预防性维护?

处境复查对今后的维护工作有重要影响,有助于对软件组织的高效管理。软件维护过程,如图 9-5 所示。

事实上,软件维护就是实际的软件工程的循环应用。不同的是,维护的类型不同,重点也有所不同,但整个方法没有变。总结软件维护过程,一般涉及以下十二项工作内容:

(1) 评价对系统的提升要求。根据软件功能和使用环境的分析,对用户提升系统的请求进行评价,评价后提出提升建议。

(2) 评价改正问题的请求。分析系统在用户使用环境中出现的问题和请求,评价后提出改正问题请求的方法。

(3) 程序紧急排错。对出现故障的程序实施紧急排错,使程序尽快恢复正常工作。

(4) 指定系统维护更新计划。根据用户的请求和系统提升建议,制定系统更新计划,确定优先级别和维护更新版本日期。

(5) 维护更新版本需求分析。详细地分析与系统更新版本有关的需求,编写出版维护更新版本的需求文档。

(6) 维护更新版本的设计。设计维护更新版本的程序,对数据结构完成版本的概要设计和详细设计。

(7) 维护更新版本的编写和测试。对正常维护的更新版本进行编码新版本的概要设计和详细设计。

(8) 新版本的发布。

(9) 实行预防性维护。对投入市场后的软件进行监督,及时掌握运行情况,如确实必要,可适当地对系统进行预防性维护,使软件处于最佳运行状态。

(10) 人员培训。针对拥护和市场需要编写维护更新版本培训资料,组织员工培训,提高员工的能力,支持新版本发布后的用户服务工作。

(11) 周期性系统评估。软件开发维护单位主动对软件进行一种定期评估,用来考察本系统开发的软件产品的效能和适用性,每次评估后应撰写评估报告。

(12) 进行执行后评审。在软件使用相当长时间后,对系统的功能和性能进行全面地和深入地评审,评估后应撰写执行后的评审报告。

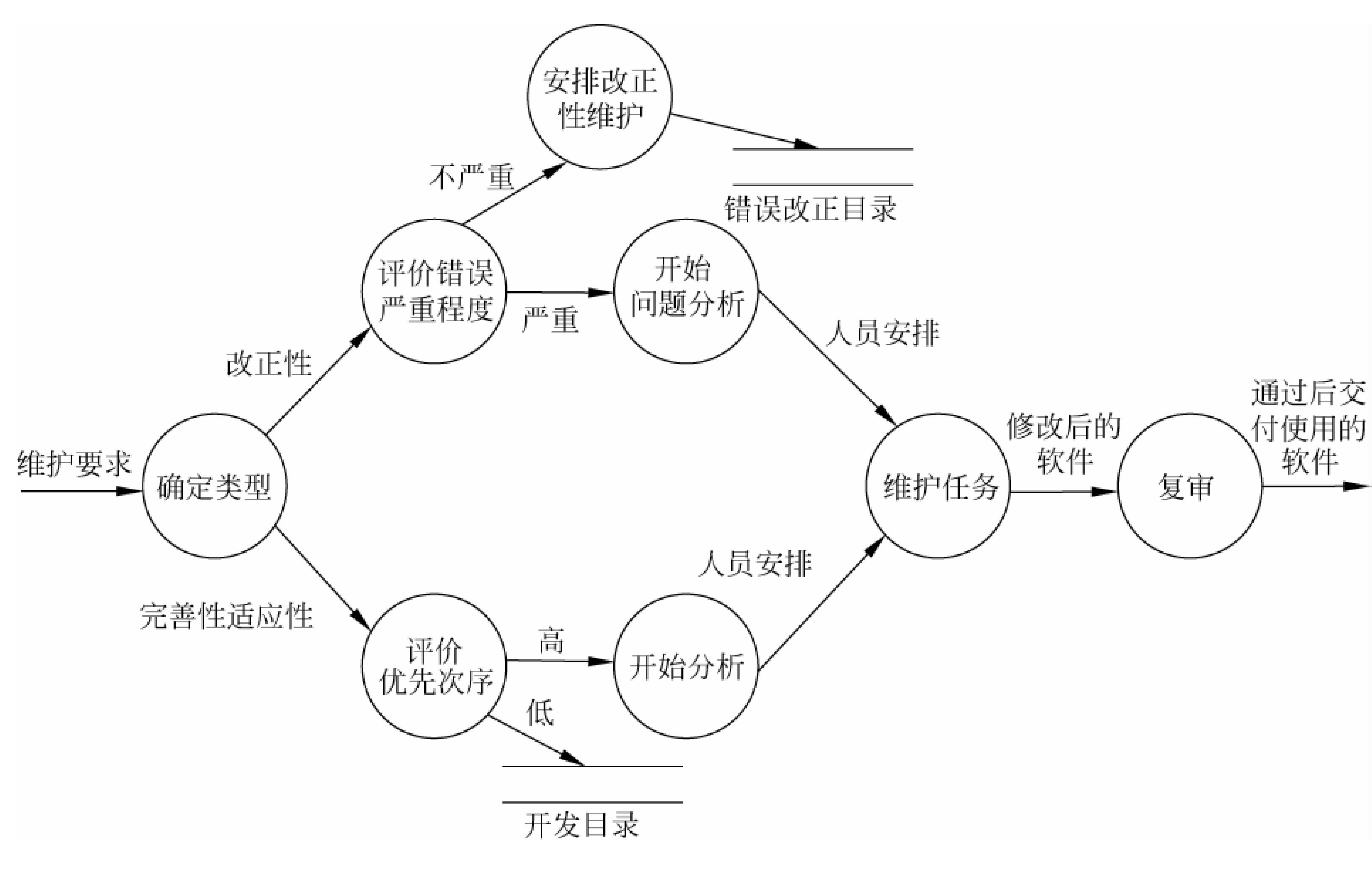


图 9-5 软件维护过程

4. 维护记录

为了更好地评估软件维护的有效性、确保软件质量、统计软件维护实际开销,应该采集每项维护工作过程中的相关数据,做好维护档案记录、构建维护数据库的工作。需要详细记录的数据包括:

- 程序标识；
- 源语句数；
- 机器指令条数；
- 所用的程序设计语言；
- 程序安装的日期；
- 自从安装之日起程序运行的次数；
- 自安装之日起程序失败的次数；
- 程序变动的层次和标识；
- 因程序变动而增加或删除的源语句数；
- 每处改动所消耗的人时数；
- 程序改动的日期；
- 软件工程师的姓名；
- 维护申请报告表的标识；
- 维护类型；
- 维护开始和结束的日期；
- 用于本次维护的人时数的累计数字；
- 完成本次维护的纯利润。

5. 维护评估

详细的维护记录可为定量评估维护工作提供有效的参考依据，以建立定量度量模型。根据评估结果，可以做出关于开发技术、语言选择、维护工作量规划、资源分配及其他许多方面的决定，并且可以利用这样的数据去分析评价维护任务。

一般来说，可以从以下七个方面来评价维护工作：

- 每次程序运行时的平均失效次数；
- 用于每一类维护活动的总人时数；
- 平均每个程序、每种语言、每种维护类型所做的程序变动数；
- 在维护过程中，增加或删除每条源程序语句花费的平均人时数；
- 用于维护每种语言所花费的平均人时数；
- 一张维护申请报告表的平均处理时间；
- 各类型维护所占的百分比。

9.4.4 软件维护的副作用

软件维护的目的是保证软件质量，延长软件使用寿命，从而提高软件的使用价值。一般地，经过一段时间的维护，软件系统中的错误会明显减少，软件的功能得到增强。但是，软件维护是一件很危险的工作，对一个复杂的逻辑过程，哪怕做一项微小的改动，都可能引入潜在的错误和缺陷。这种因软件维护或在软件维护过程中其他一些不期望的行为而引入的错误，被称为软件维护的副作用。虽然完整一致的文档资料和细致的回归测试有助于消除错误，但仍无法避免维护副作用的产生。

维护的副作用大致可分为三类：代码副作用、数据副作用和文档副作用。

1. 代码副作用

使用任何一种程序设计语言开发系统，都有可能在软件维护过程中引入不可预知的错误。特别是在软件维护人员不熟悉系统整体架构的情况下，对一条简单语句的小修改，有时也可能会带来灾难性的后果。尽管不是所有的副作用都会导致出现非常严重的后果，但修改可能导致错误，而错误会导致发生各种问题。可能产生副作用的代码修改操作有：

- 修改或删除子程序；
- 修改或删除语句标号；
- 修改或删除标识符；
- 修改代码的时序关系；
- 修改变量的存储大小；
- 修改文件的 open 或 close 操作；
- 修改逻辑运算符；
- 由设计变动引起的代码修改；
- 修改边界条件。

为保证修改代码没有引入新的错误，应进行严格地回归测试。一般情况下，修改代码所带来的副作用，可以在回归测试的过程中发现并纠正。代码副作用的范围，包括从回归测试期间发现并纠正的错误，到软件运行期间引起软件故障的问题。

2. 数据副作用

在软件维护过程中，经常要对数据结构的个别元素或结构本身进行修改。当数据结构被改变时，有可能造成软件设计与数据结构的不匹配，从而导致软件出错。数据副作用是不严谨地修改软件系统的信息结构所导致的结果。

数据副作用经常发生在一些与数据相关的修改过程中，例如：

- 重新定义局部变量和全程变量；
- 重新定义记录格式或文件格式；
- 增减数据或其他复杂数据结构的大小；
- 修改全局数据；
- 重新初始化控制标志或指针；
- 重新排列 I/O 或子程序参数表；
- 重新定义或改变接口的参数。

以上情况容易导致设计与数据不相容的错误。数据副作用可以通过详细完善的设计文档加以控制，此类文档描述了软件数据结构，并提供了一种把数据元素、记录、文件以及其他数据结构与软件模块联系起来的交叉对照表。

3. 文档副作用

软件维护应统一考虑整个软件配置，而不仅仅是源代码。如果对可执行程序的修改情况没有反映到设计文档和用户手册中，就会产生文档副作用。

对数据流、软件架构、模块接口、模块内的运算逻辑或任何其他有关特性进行修改时，都必须同时对相关技术文档进行相应修改，否则就会导致文档与程序功能不匹配、默认条件改变、新信息不正确等错误，使得文档不能反映软件当前的状态。因此，必须在软件交付之前对整个软件配置进行评审，以减少文档副作用。

应当明确：

- 对软件的任何修改都必须在相应的技术文档中反映出来，如果设计文档不能与软件当前的状况对应则比没有文档更糟；
- 对用户来说，若使用说明中未能反映修改后的状况，那么用户在这些问题上必定会出错；
- 一次维护完成之后，在再次交付软件之前应仔细复审整个配置，以有效地减少文档副作用；
- 某些维护申请不必修改软件设计和源代码，只需要指出在用户文档中不够明确的地方，整理用户文档便可达到维护的目的。

为了控制文档副作用，在维护中应做到以下几点：

- 按模块把修改分组；
- 自顶向下地安排被修改模块的顺序；
- 每次修改一个模块；
- 对于每个修改了的模块，在安排修改下一个模块之前，都要确定这个修改的副作用，可以使用交叉引用表、存储映像表、执行流程跟踪等。

9.4.5　软件可维护性

软件可维护性是指导软件开发阶段各个时期工作的一条基本原则，也是软件工程追求的目标之一。许多软件的维护十分困难，原因在于这些软件文档不齐全、质量差、开发过程中不注意采用好的方法，忽视程序设计风格等。还有一些维护要求并不是因为程序中出错提出的，而是为了适应环境变化或需求变化提出的。为了让软件能够易于维护，必须考虑使软件具有可维护性。

软件可维护性是指纠正软件系统出现的错误和缺陷，以及为满足新的要求进行修改、扩充或压缩的难易程度(维护人员理解、改正、改动和改进软件的难易程度)。

可维护性是软件产品的一个重要质量特性，是衡量软件质量的重要标准。软件生命周期每个阶段的工作都与软件可维护性有密切的关系，提高软件可维护性是软件开发各个阶段的关键目标之一。此外，软件的可维护性也是降低维护成本的重要因素，可以通过提高软件的可维护性来降低软件维护的困难程度，以控制成本。

影响软件可维护性的因素有：可理解性、可使用性、可修改性、可测试性、可移植性、可重用性、效率和可靠性。现阶段广泛使用这八个特性来度量程序的可维护性，用以了解软件是否满足了规定的维护性要求，有助于及时发现维护性的设计缺陷，还可以作为更改设计或维护安排的依据，从而指导软件维护性的分析和设计。这些特性一般表现在软件系统的许多方面，因此对于不同类型的维护，这八种特性的侧重点也有所不同，如表 9-2 所示。

表 9-2　八种可维护特性在各类维护中的侧重点

	改正性维护	适应性维护	完善性维护
可理解性	✓		
可使用性		✓	✓
可修改性	✓	✓	
可测试性	✓		
可移植性		✓	
可重用性		✓	✓
效率			✓
可靠性	✓		

1. 可理解性

软件的可理解性是指人们通过阅读源代码和相关文档，理解软件的结构、接口、功能和运行的难易程度。

对可理解性的度量，主要是对软件维护人员进行故障原因分析，或定位需要修改部分，所付出努力的程度或投入资源数量进行度量。一个可理解的软件应具备以下特性：模块化；编码风格一致；代码清晰易懂；使用有意义的数据名和过程名；结构化设计；内部文档完善；使用良好的编码工具等。

2. 可使用性

从用户角度来看，可使用性被定义为软件方便、实用和易于使用的程度。一个可使用的软件系统应该是易于操作的、能允许用户在一定程度上出错和改变，并尽可能让用户在使用软件时感到方便、舒适，不会陷入混乱状态。

软件可使用性的度量标准为：

- 软件是否具有自描述性？
- 软件是否让用户对数据处理有一个满意的和适当的控制？
- 软件是否能够被用户较容易地学会使用？
- 软件是否使用数据管理系统来自动地处理事务性工作和管理格式化、地址分配及存储器组织？
- 软件能否按照用户的要求一直保持正常的运行状态？
- 软件是否具有容错性？
- 软件是否灵活？

3. 可修改性

软件的可修改性表明了程序容易修改的程度。一个可修改的程序应当具备以下特性：

- 可理解；
- 通用，指程序适用于各种功能变化而不需要被修改；
- 灵活，指能够很容易地对程序进行修改；
- 简单。

对可修改性的度量主要是对软件系统及维护人员实现软件修改所付出的努力程度进行度量。测试可修改性的一种定量方法是修改练习，其基本思想是通过做几个简单的修改，来评价修改的难度。

设 C 是程序中各个模块的平均复杂度，n 是必须修改的模块数，A 是要修改的模块的平均复杂度。则修改的难度 D 由下式计算：

$$D = (A / C) \times n$$

4. 可测试性

软件的可测试性表明验证程序正确性的难易程度。因为程序越简单，证明其正确性就越容易。而且设计出有效的、合适的测试用例，取决于对程序的全面理解。所以，一个可测试的程序应当是可理解的、可靠的、简单的。此外，良好的软件结构、可用的测试工具和调试工具，及以前设计的测试过程也都是非常重要的。维护人员可以使用开发阶段用过的测试方案进行回归测试，而在设计阶段也应该尽力把软件设计成容易测试和容易诊断的。

可测试性度量标准如下：

- 软件是否实现了模块化？
- 软件是否结构良好？
- 软件是否可理解？
- 软件是否可靠？
- 软件是否能显示任意中间结果？
- 软件是否能以清楚的方式描述它的输出？
- 软件是否能及时地按照要求显示所有的输入？
- 软件是否有跟踪及显示逻辑控制流程的能力？
- 软件是否能从检查点再启动？
- 软件是否能显示带说明的错误信息？

对于程序模块来说，可以用程序复杂度来度量其可测试性。模块的环形复杂度越大，可执行的路径就越多，因此，全面测试它的难度就越高。

5. 可移植性

软件的可移植性是指，把程序从一种计算环境（硬件配置和操作系统）转移到另一种计算环境的难易程度。一个可移植的软件应具有结构良好、灵活、不依赖于某一具体硬件环境或操作系统的性能。

主要从以下方面进行可移植性的度量：

- 软件是否使用独立于特定机器的高级语言编写？
- 软件是否采用广泛使用的标准化的程序设计语言来编写程序？是否仅使用了这种语言的标准版本和特性？
- 程序中是否使用了标准函数库功能和子程序？
- 程序中是否极少使用或根本不使用操作系统的功能？
- 程序在执行之前是否初始化内存？
- 程序在执行之前是否测试当前的 I/O 设备？

- 程序是否把与机器相关的语句分离了出来，集中放在了一些单独的程序模块中，并有说明文件？
- 程序是否是结构化的？并允许在规模小一些的计算机上分段（覆盖）运行？
- 程序中是否避免采用了依赖于字母数字或特殊字符的内部位表示？

6. 可重用性

如果软件中的某些部分不做修改或稍加修改就可以在不同环境中多次重复使用，则认为其具有可重用性。大量使用可重用的软件构件来开发软件，可以从下述两个方面提高软件的可维护性。

- 通常，可重用的软件构件在开发时都会经过很严格地测试，可靠性比较高，且在每次重用过程中都会出现并清除一些错误，随着时间推移，这样的构件将变成实质上无错误的。因此，软件中使用的可重用构件越多，软件的可靠性越高，改正性维护需求就越少。
- 很容易修改可重用的软件构件使之再次应用在新环境中，因此，软件中使用的可重用构件越多，适应性和完善性维护也就越容易。

7. 效率

效率表明一个软件能执行预定功能而又不浪费机器资源（包括内存容量、外存容量、通道容量和执行时间）的程度。

效率度量标准如下：

- 软件是否实现了模块化？结构是否良好？
- 程序是否消除了无用的标号与表达式，以充分发挥编译器优化作用？
- 程序的编译器是否有优化功能？
- 是否把特殊子程序和错误处理子程序都归入单独的模块中？
- 是否以快速的数学运算代替了较慢的数学运算？
- 是否尽可能地使用了整数运算，而不是实数运算？
- 是否在表达式中避免了混合数据类型的使用，消除了不必要的类型转换？
- 程序是否避免了对非标准的函数或子程序的调用？
- 在几条分支结构中，是否最有可能为“真”的分支首先得到测试？
- 在复杂的逻辑条件中，是否最有可能为“真”的表达式首先得到测试？

8. 可靠性

软件的可靠性表明一个程序按照用户的要求和设计目标，在给定的一段时间和规定的条件下，软件维持正确运行的概率，以及发生故障后，软件系统重新恢复其性能水平和直接受影响数据的难易程度。

对于可靠性，度量标准有：

- 平均失效间隔时间 MTTF
- 平均修复时间 MTTR
- 有效性 A（A＝ MTBD/（MTBD＋MDT））

度量可靠性的方法有：

- 根据程序错误统计数字，进行可靠性预测。常用方法是利用一些可靠性模型，根据程序测试时发现并排除的错误数预测平均失效间隔时间 MTTF。
- 根据程序复杂性，预测软件可靠性。用程序复杂性预测可靠性，前提条件是可靠性与复杂性有关。因此可用复杂性预测出错率。程序复杂性度量标准可用于预测哪些模块最可能发生错误，以及可能出现的错误类型。

9. 其他间接定量度量可维护性的方法

还有一些与软件维护期间工作量有关的数据，通过它们可以间接地对软件可维护性做出估计。

- 问题识别的时间；
- 分析、诊断问题的时间；
- 局部测试的时间；
- 修改规格说明的时间；
- 因管理活动拖延的时间；
- 收集维护工具的时间；
- 具体的改错或修改的时间；
- 集成或回归测试的时间；
- 维护的评审时间；
- 恢复时间。

这些数据反映了维护全过程中检错-纠错-验证的周期，即从检测出软件存在的问题开始至修正它们并经回归测试验证这段时间。可以粗略地认为，这个周期越短，维护越容易。

9.4.6 可维护性复审

为了使软件具备可维护性，需要在软件开发的各个阶段采取相应的措施加以保证。在软件工程的每一个阶段，都必须考虑并努力提高软件的可维护性，在每个阶段结束前的技术审查和管理复审中，应该着重对可维护性进行复审。

(1) 在需求分析阶段的复审过程中，应该对将来要改进的部分和可能会修改的部分加以注意并指明；应该讨论软件的可移植性问题，并且考虑可能影响软件维护的系统界面。

(2) 在软件设计的复审中，应从容易修改、模块化和功能独立的目标出发，评价软件的结构和过程；从软件质量的角度全面评审总体设计、过程设计、数据设计和界面设计。另外，还应对将来可能修改的部分预做准备。

(3) 代码复审中，应强调编码风格和内部说明文档这两个影响可维护性的因素。设计和编码阶段都应尽量使用可重用的软件构件，在开发新构件时，也应该注意提高构件的可重用性。

(4) 所有的测试工作都暗示着软件在正式交付使用前，可能需要进行预防性维护的部分。测试结束时将进行最正式的可维护性复审，称为配置复审。其目的是保证软件配置的所有成分是完整的、一致的和可理解的，同时也为了便于修改和管理已经编目归档的文件。

在完成了每项维护工作之后，都应该对软件维护本身进行认真仔细地复审。而且维护应该是针对整个软件配置的，而不仅仅局限于修改源程序代码。当对源程序代码的修改没有反映在设计文档或用户手册中时，则会产生严重的后果。如果在软件维护结束，再次交付用户使用之前，对软件配置进行严格地复审，则可减少文档的问题，提高软件的可维护性。

9.4.7 提高软件的可维护性

1. 规定软件维护优先级

影响软件可维护性的因素中，有的特性是相互促进的，例如可理解性和可测试性、可理解性和可修改性。但另一些特性却是相互抵触的，例如效率和可移植性、效率和可修改性等。因此，尽管可维护性要求的每一种特性都能得到满足，但它们的相对重要性随程序的用途及计算环境的不同而改变。例如，对编译程序来说，可能强调效率；但对管理信息系统来说，则可能强调可使用性和可修改性。所以，在对软件的可维护性特性提出目标的同时，还应当规定其优先级，以侧重不同软件的维护要求，提高软件可维护性。

2. 使用提高软件质量的技术和工具

采用提高软件质量的技术和工具，如采用面向对象、软件重用等先进的开发技术，也可有效提高软件的可维护性。常用的方法有以下几种。

1）模块化

模块化的优点是模块的独立性特征。如果要改变某个模块的功能，只需要改动该模块便可完成，对其他模块的影响很小；如果需要增加新功能，则只添加完成这些功能的新模块或模块层即可；程序的测试与重复测试会比较容易；程序错误也易于定位和修正。

2）结构化程序设计

可采用结构化程序设计方法使得模块结构及模块间的相互作用标准化，获得良好的程序结构，提高软件的可维护性。

3）采用备用件方法

当要修改某个模块时，用一个结构良好的新模块进行替换。这种方法提供了一个用结构化模块逐步替换非结构化模块的机会，有利于减少新的错误，并且仅要求了解所替换模块的外部接口特性，而不需要了解其内部工作情况。

4）采用自动重建结构和重新格式化的工具（结构更新技术）

该方法采用如代码评价程序、格式重定程序、结构化工具等自动软件工具，把非结构化代码转换成良好结构代码。使用这种方法产生的结构化程序，执行过程与结构化之前的源程序是一样的，都对相同的数据执行相同的操作顺序，而源程序中的逻辑错误也会继承下来。程序结构化转换的过程步骤如下：确保程序编译没有语法错误；借助结构化工具，重新构造源代码；利用重定义格式程序进行缩进和分段；利用优化编译器重新编译代码，提高程序效率。

5）改进现有程序不完善的文档

建立或补充系统说明书、设计文档、模块说明数以及在源程序中插入必要的注释。

6）维护过程结构化

提高现有系统可维护性的一个比较好的方法是使维护过程结构化，而不是使现有系统重新结构化。

3. 进行明确的质量保证审查

进行明确的质量保证审查对于软件可维护性的提高，是一项很有用的技术。审查用来检测在开发和维护阶段内发生的质量变化，一旦检测出问题，就可以采取措施纠正，以控制不断增长的软件维护成本，延长软件系统的有效生命周期。

软件质量保证审查的类型包括以下四个。

1）在检查点进行复审

在软件开发的最初阶段就应把质量要求考虑进去，并把开发过程中各阶段的终点设置为检查点进行复审，检查已开发的软件是否符合标准，是否满足规定的质量需求。在不同的检查点，检查的重点也有所不同，如图 9-6 所示。

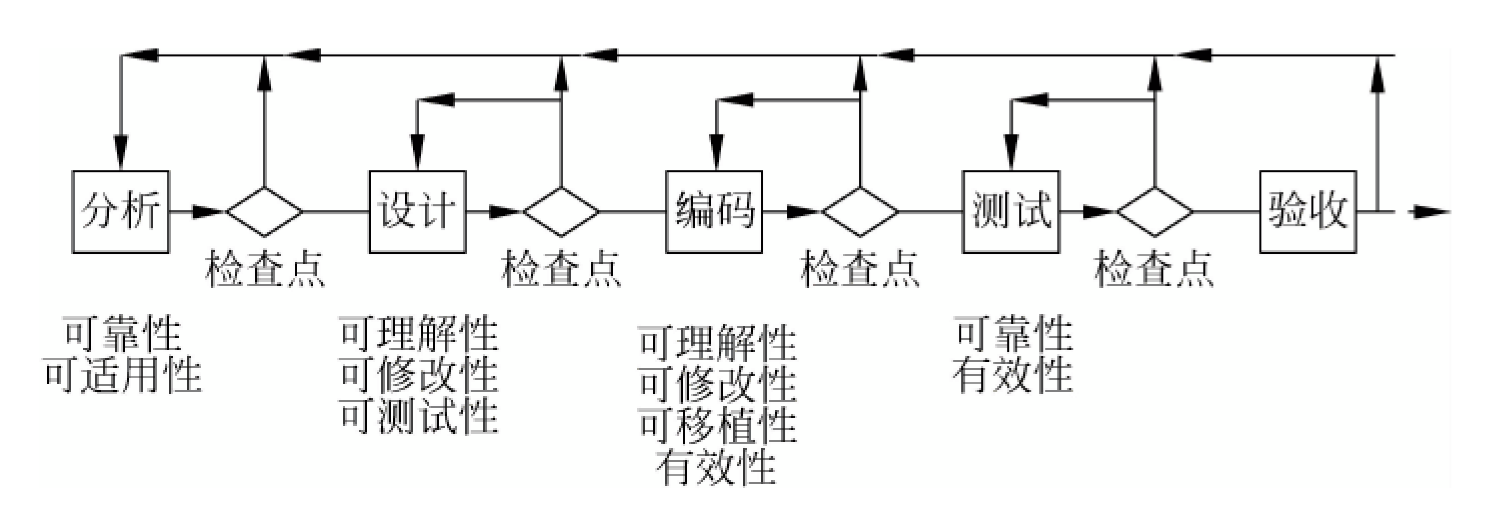


图 9-6　检查点复审

进行检查点复审时，可以使用各种质量特性检查表，或用度量标准来检查可维护性。各种度量标准应当在管理部门、用户、软件开发人员、软件维护人员当中达成一致意见。

2）验收审查

验收审查从维护的角度提出软件验收的条件和标准，是软件交付使用前的最后一次检查，属于验收测试的一部分，也是软件投入运行之前保证可维护性的最后机会。

下面是验收审查的几个验收标准。

- 需求和规范标准：需求应当以可测试的术语进行书写，排列优先次序和定义；区分必需的、任选的、将来的需求；包括对系统运行时的计算机设备的需求；对维护、测试、操作，以及维护人员的需求；对测试工具等的需求。
- 设计标准：程序应设计成分层的模块结构。每个模块应完成唯一的功能，并达到高内聚、低耦合；通过一些可知预期变化的实例，说明设计的可扩充性、可缩减性和可适应性。
- 源代码标准：尽可能使用最高级的程序设计语言，且只使用语言的标准版本；所有的代码都必须具有良好的结构；所有的代码都必须文档化，在注释中说明其输入、输出，以及便于测试、再测试的一些特点与风格。
- 文档标准：文档中应说明程序的输入、输出；使用的方法、算法；错误恢复方法；所有参数的范围及默认条件等。

3）周期性维护审查

与硬件的定期检查一样，软件的周期性维护审查可以及时跟踪软件质量的变化。

周期性维护审查实际上就是开发阶段检查点复审的继续，并且采用的检查方法、检查内容都是相同的。对现有软件系统进行周期性地维护审查，将审查结果与以前的维护审查结果、验收审查结果、检查点检查结果相比较，任何一种改变都表明在软件质量或其他类型的问题上可能起了变化。及时对发生改变的原因进行分析处理，可提高软件的可维护性。

4）对软件包进行检查

软件包是一种标准化的，可为不同单位、不同用户使用的软件。由于一般不会向用户提供软件包的源代码和程序文档，所以对软件包的维护采取了以下方法。

- 检查人员或维护人员要仔细分析、研究卖主提供的用户手册、操作手册、培训教程、新版本说明、计算机环境要求书以及卖方提供的验收测试报告等。在此基础上，深入了解本单位的希望和要求，编制软件包的检验程序。
- 维护人员既可以利用卖方提供的验收测试实例，也可以自行设计新的测试实例。根据测试结果，检查和验证软件包的参数的控制结构，以完成软件包的维护。

4. 采用可维护性的程序设计语言

程序设计语言的选择，对程序的可维护性影响很大，如图 9-7 所示。

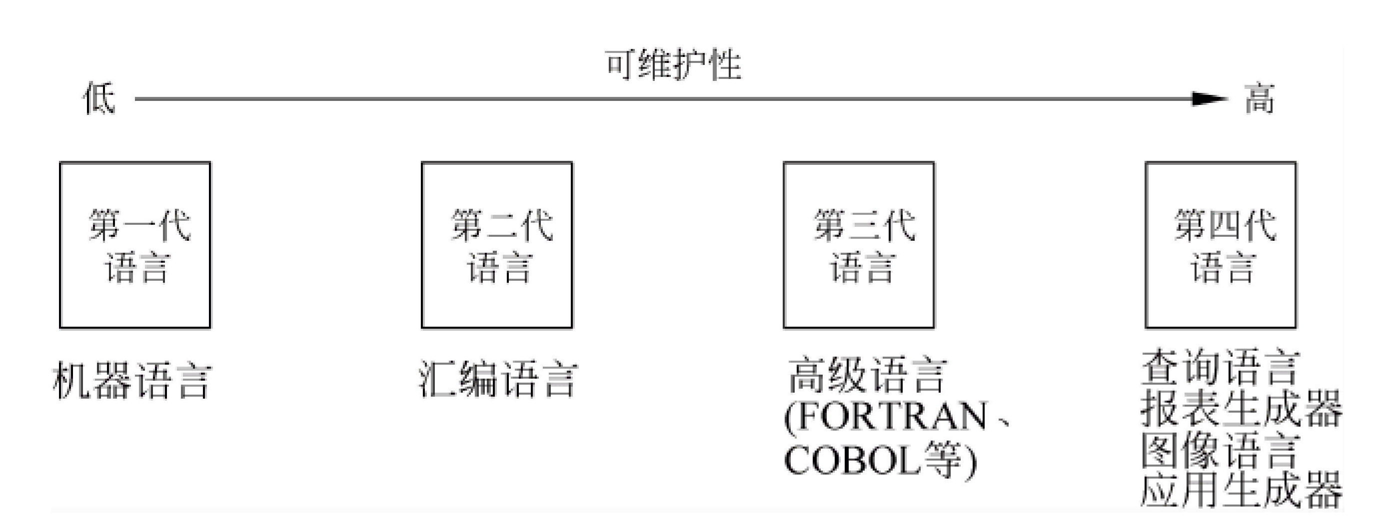


图 9-7 程序设计语言可维护性

第四代语言包括查询语言、报表生成器、图像语言和应用生成器。不论采用何种过程化或非过程化的第四代语言，编写出的程序都应容易被理解和修改，而且其产生的指令条数可能要比用第二代或第三代语言编制出程序的少一个数量级，但开发速度却快许多倍。有些非过程化的第四代语言，用户甚至不需要指出实现的算法，仅需要向编译程序或解释程序提出自己的要求，由编译程序或解释程序做出实现用户要求的智能假设。总之，从维护角度来看，第四代语言比其他语言更容易维护。

5. 改进文档

即使是一个十分简单的程序，要想有效地、高效率地维护它，也需要编制文档来解释其目的及任务。文档作为对软件总目标、软件各组成部分之间的关系、软件设计策略、软件实现过程的历史数据等的说明和补充，对提高软件可维护性有着非常重要的作用。

改进软件文档，采用简洁、一致的风格；在程序中插入注释以提高程序的可理解性；以移行、空行等明显的视觉组织方法来突出程序的控制结构，都将简化维护工作、提高程序的可维护性。

9.5 软件维护的最新方法

传统的软件维护方法，是采用类似软件开发过程的方式修改软件，以改正错误或满足新的需要，从而延长现有软件的生命周期。但是，随着计算机技术和软件应用环境的快速发展，仅靠维护阶段的修补改进工作，很难使旧的软件系统跟上变化的步伐，满足用户需求。而且，软件运行时间越长，维护的难度就越大，成本也会迅速增加，甚至会超过开发一个新软件的总成本。因此，必须采用新的软件维护方法，使得原有软件能够充分发挥作用，同时又可避免维护的成本过高。

9.5.1 软件的逆向工程和再工程

逆向工程(Reverse Engineering)源于商业及军事领域中的硬件分析，是一种产品设计技术再现过程，即对一项目标产品进行逆向分析及研究，从而演绎并得出该产品的处理流程、组织结构、功能特性、技术规格等设计要素，以制作出功能相近，但又不完全一样的产品。软件的逆向工程与之类似，即通过分析程序恢复设计结果，是一个从现存的程序代码中抽取数据结构、体系结构和程序设计信息，以便在比源代码更高抽象层次上建立程序表示的过程。

再工程(Re-engineering)的概念起源于传统工程业界，是指为摆脱旧的组织和管理业务的规则，使用新的现代技术从根本上重新设计业务过程，以使它们性能得到极大改善。再工程是一项针对工程业务过程和管理的技术革命。软件再工程是指对既存对象系统进行调查，并将其重构为新形式代码的开发过程，最大限度地重用既存系统的各种资源是再工程的最重要特点之一。

软件再工程可以看做是将新技术、新工具应用于旧软件的一种较彻底的预防性维护，是目前预防性维护所采用的主要技术，是为了以新形式重构已存在软件系统而实施的检测、分析、更替以及随后构建新系统的工程活动。软件再工程的目的是理解已存在的软件，然后对该软件重新实现以期增强其功能，提高性能，或降低实现难度，客观上达到维持软件的现有功能并为今后加入新功能做好准备的目标。

9.5.2 逆向工程

逆向工程通过反汇编和调试等方法分析计算机程序的可执行代码，可以从一个非结构化的无文档的源代码或目标代码中提取设计信息，如图 9-8 所示。理想情况是抽象层次尽可能高，也就是说，逆向工程过程应当能够导出过程性设计的表示(最低层抽象)、程序和数据结构信息(低层抽象)、数据和控制流模型(中层抽象)和实体关系模型(高层抽象)。随着抽象层次的增加，可以给软件工程师提供更多的信息，使得程序更容易被理解。

逆向工程的核心活动是提取抽象，工程师必须评价旧程序并从源代码中抽取出被完成的处理、被应用的用户界面以及被使用的数据结构或数据库等有意义规约。

逆向工程导出的信息可分为如下四个抽象层次。

- 实现级：包括程序的抽象语法树、符号表等信息。

- 结构级：包括反映程序成分之间相互依赖关系的信息，如调用图、结构图等。
- 功能级：包括反映程序段功能及程序段之间关系的信息。
- 领域级：包括反映程序成分或程序实体与应用领域概念之间对应关系的信息。

对于一项具体的维护任务，一般不必导出所有抽象级别上的信息，如代码重构任务，只需要获得实现级信息即可。

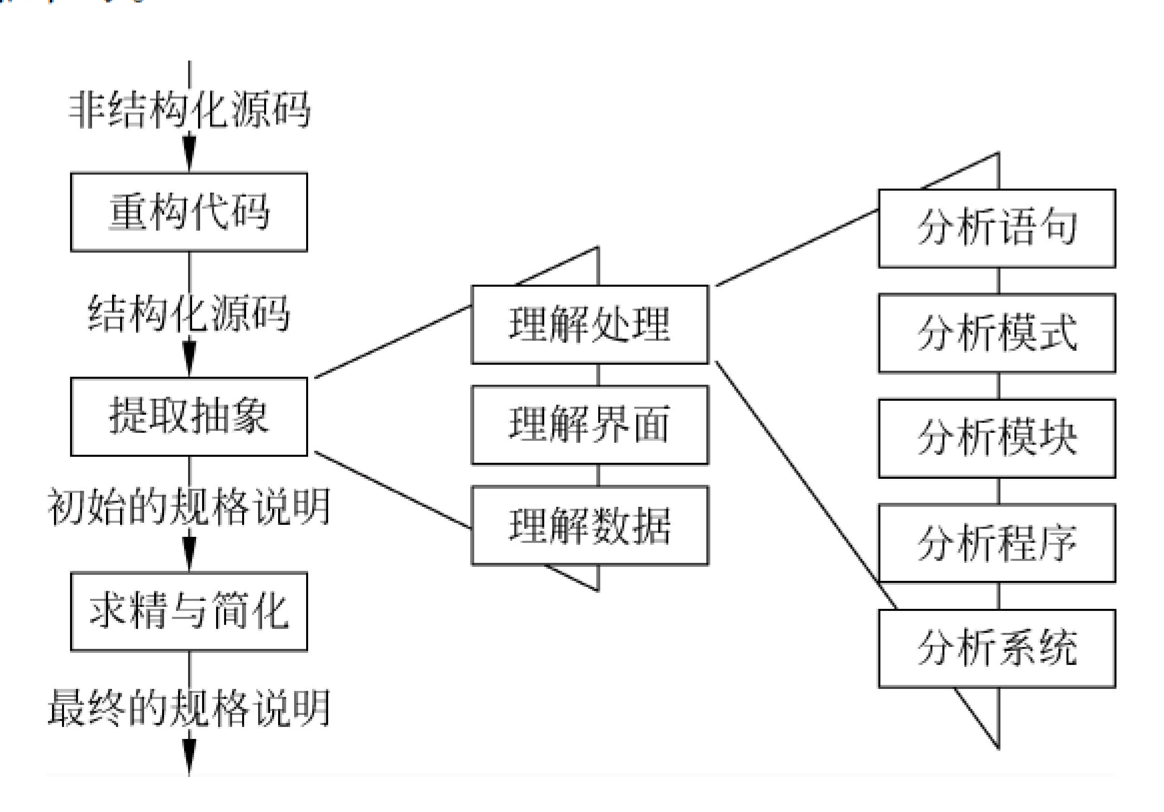


图 9-8 逆向工程过程

根据源程序的类别不同，逆向工程还可以分为以下三种。

1）对用户界面的逆向工程

现代软件一般都采用非常友好的交互界面，当准备对旧的软件进行用户界面的逆向工程时，必须先理解旧软件的用户界面，并且刻画出界面的结构和行为。

2）对数据的逆向工程

由于程序中存在许多不同类型的数据，例如内部数据结构、底层数据库和外部文件。其中，对内部数据结构的逆向工程可以通过检查程序代码以及变量来完成；而对数据库结构的逆向工程可通过建立一个初始的对象模型、确定候选键、精华实验性的类、定义一般化以及发现关联来完成。

3）对理解的逆向工程

为了理解过程的抽象，代码的分析必须在不同的层次进行。对于大型系统，逆向工程通常用半自动化的方法来完成。

9.5.3 再工程

软件再工程是一类软件工程活动，与软件开发相比，软件再工程不是从编写规格说明开始，而是从原有的软件着手开发，是一个将逆向工程、重构和正向工程组合起来，把现存系统重新构造为新形式的工程过程，通过再工程，能获得可维护性好的新软件。系统理解是再工程的基础，包括对系统运行、源代码、设计、分析、文档等的全面理解。在理解的基础上，执行重构生成一个设计，产生与原软件相同的功能，但具有比原软件更高的质量。

1. 实施软件再工程的意义

- 再工程可帮助软件机构降低软件演化的风险。改进原有软件时必须频繁地对软件

实施变更，降低软件的可靠性，而软件再工程可以降低变更带来的风险。

- 再工程可帮助软件机构补偿软件投资。许多软件机构每年要花费大量的资金用于开发软件。如果采用再工程，而不是完全舍弃原软件，可以部分补偿他们在软件上的投资。
- 再工程可使得软件易于进一步变更。再工程可使程序员更容易理解程序，更容易对其开展工作，从而提高维护工作的生产效率。
- 软件再工程有着广阔的市场。
- 再工程是推动软件自动维护的发展动力。

2. 软件再工程过程模型

典型的软件再工程过程模型定义了六类活动。如图 9-9 所示的再工程模型，意味着其任意组成部分的每个活动都可能被重复，而且对于任意一个特定的循环来说，过程可以在完成任意一个活动之后终止。并且在某些情况下，这些活动也并非以线性顺序发生，可能会产生交错情况。例如，为了理解某个程序的内部工作原理，可能在文档重构开始之前先进行逆向工程。

下面逐一介绍软件再工程过程模型中定义的六类活动。

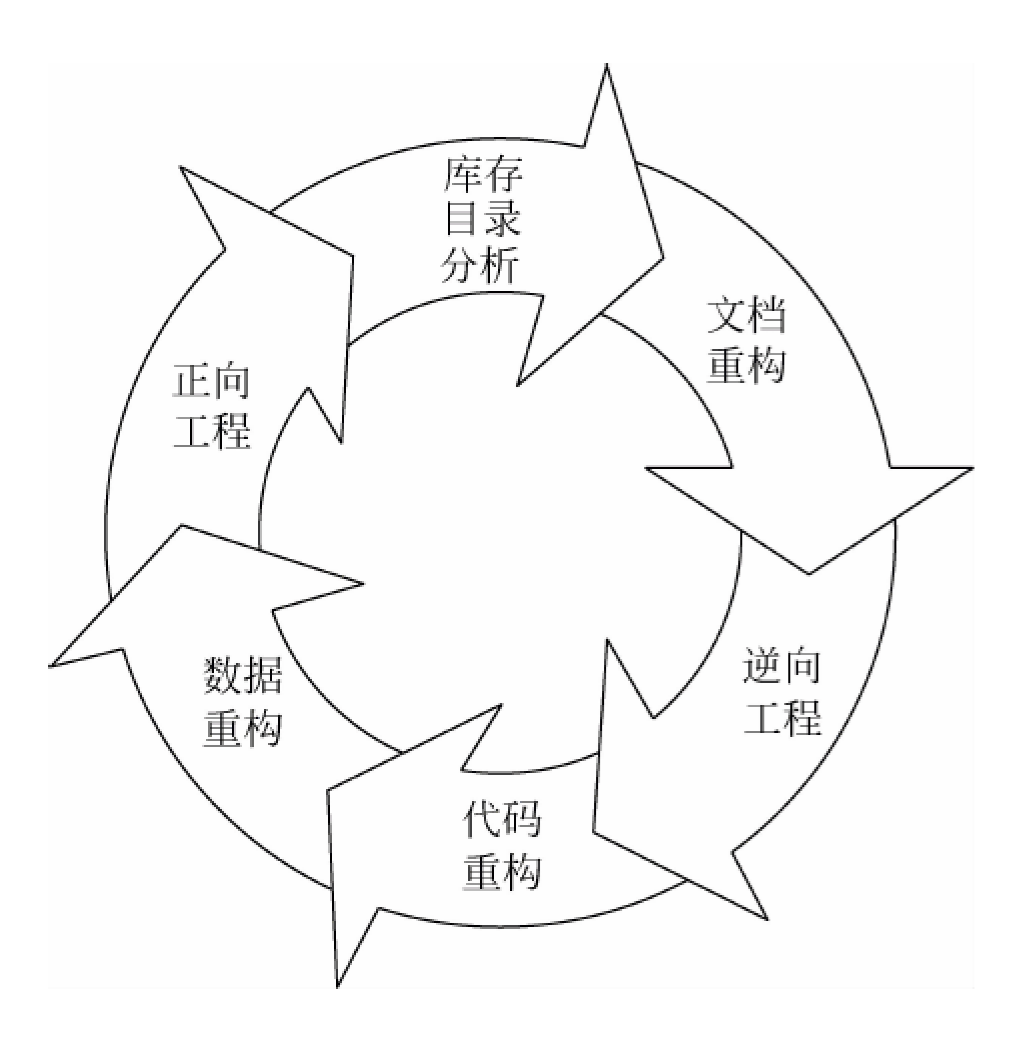


图 9-9 再工程过程模型

1）库存目录分析

每个软件组织都应该保存其拥有的、所有应用系统的库存目录。该目录包含关于每个应用系统的基本信息，例如：

- 应用系统的名称
- 最初构建的日期
- 已进行过的实质性修改次数
- 完成修改花费的总劳动
- 驻留的系统
- 和该系统有接口的其他应用
- 访问的数据库
- 过去 18 个月报告的错误
- 用户的数量
- 安装该系统的机器数量
- 程序结构的复杂度、代码复杂度和文档复杂度
- 文档质量
- 整体可维护性等级
- 预期寿命
- 未来 36 个月内的预期修改次数
- 年度维护成本
- 业务重要程度

上述目录应该定期整理修订，应用的状况（如业务重要程度）可能随时间发生变化，其结

果是再工程的优先级将发生变化。在库存目录分析阶段，应对每一个现存软件系统采集上述信息并通过局部重要标准对其排序，根据优先级不同选出再工程的候选软件，进而合理分配资源。

每一个大的软件开发机构都拥有上百万行老代码，除去不频繁使用且不需要改变的程序外，它们都可以是逆向工程和再工程的对象。但是，逆向工程和再工程工具尚不成熟，目前仅能对有限种类的应用系统执行逆向工程或再工程，而且代价十分高昂，因此，对库中每个程序都做逆向工程和再工程是不现实的。下述三类程序有可能成为预防性维护的对象：

- 预定将要使用多年的程序
- 当前正在成功地使用着的程序
- 在最近的将来可能要做重大修改或增强的程序

2）文档重构

老化软件最大的问题的是缺乏有效文档，软件再工程试图重建文档。由于文档重构是一项非常耗时的工作，所以开发人员没有必要重建所有文档，而应该采取“使用则建”的原则，尽量将文档重建工作量降低到最低。

针对不同情况，文档重建的处理方法如下：

- 建立文档非常耗费时间，不可能为数百个程序都重新建立文档。如果一个程序是相对稳定的，正在走向其生命的终点，而且可能不会再经历什么变化，则应当保持现状。
- 为了便于今后的维护，文档必须更新，但是由于资源有限，应采用“使用时建文档”的方法。也就是说，不是一下子把某应用系统的文档全部都重建起来，而是只针对系统中当前正在修改的部分建立完整文档。随着时间的推移，文档将逐步实现完整有效。
- 如果某应用系统是完成业务工作的关键，而且必须重构全部文档，则仍应该设法把文档工作减少到必需的最小量。

3）逆向工程

逆向工程是一个对已有系统分析的过程，通过分析识别出系统中的模块、组件及它们之间的关系，并以另一种形式或在更高的抽象层次上，创建出系统表示。逆向工程的目的就是在缺少文档说明、根本没有文档的情况下，还原出软件系统的设计结构、需求实现，并尽可能地找出内部的各种联系、相应的接口等，从而恢复已遗失的信息，侦测出存在的缺陷，生成可变换的系统视图，综合出较高的抽象表示。

4）代码重构

代码重构是软件再工程中最常见的活动，其目的是重构可疑模块的代码，生成功能相同但质量更高的程序。因为某些老程序的体系结构比较完整、合理，但是个别模块的编码方式却是难以被理解、测试和维护的。

通常，重构并不修改软件的整个体系结构，仅关注个体模块的内部设计细节和局部数据结构，重新改写有问题的代码，用新生成的易于理解和维护的代码替代原有的代码。如果重构扩展到模块边界之外并涉及软件体系结构，则变成了正向工程。

代码重构活动的步骤如下所示：

（1）用重构工具分析源代码，标注出与结构化程序设计概念相违背的部分。

(2) 构建有问题的代码(此项工作可自动进行)。

(3) 复审和测试重构代码(以保证没有引入异常)并更新对应文档。

5) 数据重构

软件再工程中的代码修改往往会涉及数据,并且随着需求的发展,原有的数据可能已经无法满足新的处理要求,因而需要重新设计数据结构,即对数据进行再工程。与代码重构不同,数据重构是一种发生在低抽象层次上的全范围再工程活动。大多数情况下,数据重构开始于逆向工程活动,开发人员首先进行数据分析,分解现有的数据结构,必要时定义数据模型,标识数据对象和属性,并从软件质量的角度复审现存的数据结构,去除数据中的冗余或不一致;其次,将现有的数据结构进行重新设计或扩展,以适应软件再工程的数据处理要求;最后,将现有的物理数据进行转换,迁移到新的数据存储中。

事实上,对于许多应用系统来说,数据体系结构比源代码本身对软件的生存力有更大影响。而正是由于数据体系结构对软件体系结构及程序中的算法有很大影响,对数据的修改必然会导致体系结构或代码层的改变。

6) 正向工程

当一个正常运行的软件系统需要进行结构化翻新时,就可对其实施软件再工程的正向工程。

正向工程(也称为革新或改造)应用软件工程的原理、概念、技术和方法来重新开发某个现有的应用系统,从现有程序中恢复设计信息去改变或重构现有系统,以提高其整体质量。大多数情况下,被再工程的软件既重新实现了现有系统的功能,又将新的用户需求和技术需求集成到再工程中,使新构建的系统扩展了旧系统的能力,提高了整体性能。

9.5.4　软件再工程风险

软件再工程与任何其他软件工程项目一样可能会遇到各种风险。软件管理人员必须在再工程活动之前对风险进行分析,采取适当的对策,预防风险带来的损失。

软件再工程风险主要有以下六种。

(1) 过程风险。包括过高的再工程人工成本;在规定的时间内未达到成本-效益要求;未从经济上规划再工程的投入;对再工程项目的人力投入放任自流;对再工程方案缺少管理。

(2) 人员风险。软件人员可能对再工程项目意见不一致,导致影响工作进展;程序员工作效率低。

(3) 应用风险。包括再工程项目缺少该应用领域专家的支持;对源程序体现的业务知识不熟悉;再工程项目的工作完成得不充分。

(4) 技术风险。包括恢复的信息是无用的或未被充分利用;大批昂贵的文档被开发出来;逆向工程得到的成果不可共享;采用的方法对再工程目标不适合;缺乏再工程的技术支持。

(5) 工具风险。软件人员过分依靠不成熟的工具,或未经安装的工具。

(6) 策略风险。包括对整个再工程方案的承诺不成熟;对暂定的目标无长远考虑;对程序、数据和工程过程缺乏全面的观点;无计划地使用再工程工具。

9.6 软件维护文档

软件维护文档，包括软件系统原有文档（软件文档）和维护阶段产生的新文档（维护过程文档）。由于大型软件系统在长期使用的过程中必然会经受多次修改，所以软件维护文档比程序代码更重要。

9.6.1 软件文档

软件文档是在软件工程中，对各项活动、需求、过程或结果进行描述、定义、规定、报告或认证的书面及图示信息，可以分为系统文档和用户文档两类。系统文档描述系统设计、实现和测试等各方面的内容；用户文档主要描述系统功能和使用方法，并不关心这些功能是怎样实现的。

总的说来，软件文档应该满足下述要求：

（1）必须描述如何使用这个系统，在没有这种描述时，即使是最简单的系统也无法使用；

（2）必须描述怎样安装和管理这个系统；

（3）必须描述系统需求和设计；

（4）必须描述系统的实现和测试，以便使系统成为可维护的。

1. 系统文档

系统文档可分为开发文档和管理文档两类，包括整个软件工程过程中从软件问题定义到验收测试计划，所有和系统实现有关的文档。

开发文档包括：可行性研究报告；软件需求规格说明书；数据要求说明书；总体设计说明书；详细设计说明书。

管理文档包括：项目开发计划；测试计划；测试分析报告；开发进度月报；项目开发总结报告；维护修改建议。

系统文档应该能引导读者从系统概貌进行了解，到对系统各个方面所有特点的更形式化、更具体的认识。描述系统设计、实现和测试的文档对于理解程序和维护软件来说是极其重要的。

2. 用户文档

用户一般是首先通过用户文档了解系统的，它应该能使用户获得对系统功能、性能、应用范围等系统外部特征的准确初步印象，并且其组织结构方式应便于用户根据需要阅读有关的内容。

用户文档至少应该包括下述六方面的内容。

（1）安装说明。说明怎样安装这个系统以及怎样使系统适应特定的硬件配置。

（2）功能描述。介绍系统各项功能。

（3）使用手册。简要说明如何着手使用软件，应通过实例和图示说明怎样使用常用的系统功能。

（4）异常情况处理。说明软件系统出现异常情况或用户操作错误时应怎样恢复和重新

启动。应提供售后服务电话和电子邮箱，供用户咨询使用。

（5）参考手册。详尽描述用户可以使用的所有系统设施以及它们的使用方法，还应该解释系统可能产生的各种出错信息的含义（对参考手册最主要的要求是完整，因此通常使用形式化的描述技术）。

（6）操作员指南（如果需要有系统操作员的话）。说明操作员应该如何处理使用中出现的各种情况。

9.6.2　维护过程文档

在软件维护阶段，除了及时修改原有软件文档，使其与软件产品变化同步之外，还会产生新的文档以记录软件维护的工作过程和内容，这类文档称为维护过程文档，主要有软件维护申请报告和软件修改报告两种。

1. 软件维护申请报告

即由用户填写并提交的维护请求和问题描述，如图 9-10 所示。

软件维护申请报告表			
项目名称		项目编号	
申请需求描述			
申请维护类型	□改正性维护　□适应性维护　□完善性维护		
维护要求及优先级			
预期维护结果			
申请维护安排	□远程维护　□现场维护		
申请人		申请日期	
申请审批结果	□批准　□拒绝		
评价负责人		受理日期	

图 9-10　软件维护申请报告

2. 软件修改报告

在维护工作过程中，软件组织内部应制定出一个软件修改报告，以记录维护工作相关信息，如图 9-11 所示。

软件修改报告			
项目名称		维护类型及优先级	
源程序名称		备份程序名称	
相关文档列表			
维护描述：			
日期	修改内容	修改原因	备注
增加代码行数		删除代码行数	
修改代码行数		程序注释修改	
相关文档修改		维护环境	
维护起止日期		维护人员	

图 9-11　软件修改报告

9.7 本章小结

软件产品总体可分为系统软件、支持软件和应用软件三大类。

软件产品的发布时机是由市场利润、开发进度、产品功能与质量、版本管理状态、客户可接受程度等多方面的因素决定的。严格按照软件产品发布流程发布软件版本是建立和完善软件产品版本控制，保证软件产品质量，确保应用软件正常发布的关键。

软件产品的实施，是一个软件产品从内部开发完成、产品发布，到系统正式运行之间的一个阶段过程。对于定制的行业应用软件，实施是产品开发的延续；对于大型复杂软件系统，实施是包括咨询服务在内的软件产品的组成部分。一个标准、完整的软件产品实施过程可分为六个步骤：实施准备、业务交流、软件实施培训、系统初始化、新老系统切换及试运行、系统验收。

软件维护就是在软件已经交付使用之后，为了改正软件中的错误或增加功能以适应新需求而修改软件的过程。一般不包括对软件体系结构上的重大改变，其特点是工作量大、代价高昂、问题很多。根据要求进行维护原因的不同，软件维护可以分为改正性维护、适应性维护、完善性维护和预防性维护。

维护过程实际上是简化和修改了的软件开发过程，并且软件维护有关的工作通常很早就开始进行了。首先必须建立一个维护组织，然后确定维护报告内容，记录维护流程、保存记录，最后确定评估及复审标准。此外，还应当为每个维护申请确定一个标准化的事件序列。维护的副作用大致可分为三类：代码副作用、数据副作用和文档副作用。

软件可维护性是指纠正软件系统出现的错误和缺陷，以及为满足新的要求进行修改、扩充或压缩的难易程度(维护人员理解、改正、改动和改进软件的难易程度)。影响软件可维护性的因素有：可理解性、可使用性、可修改性、可测试性、可移植性、可重用性、效率和可靠性。提高软件可维护性的方法有：规定软件维护优先级；使用提高软件质量的技术和工具；进行明确的质量保证审查；采用可维护性的程序设计语言；改进文档。

软件的逆向工程是一种产品设计技术再现过程，软件再工程可以看做是将新技术、新工具应用于旧软件的一种较彻底的预防性维护，它们是软件维护的最新方法。

软件文档是在软件工程中，对各项活动、需求、过程或结果进行描述、定义、规定、报告或认证的书面及图示信息，可以分为系统文档和用户文档两类。软件维护文档，包括软件系统原有文档和维护阶段产生的新文档。由于大型软件系统在长期使用的过程中必然会经受多次修改，所以软件维护文档比程序代码更重要。

习题 9

1. 简述软件产品可分为哪三类？
2. 简述软件产品发布的流程规范。
3. 软件产品实施过程可分为哪几个步骤？

4．什么是软件维护？软件维护的特点有哪些？
5．软件维护过程包括哪些活动？
6．影响软件可维护性的因素有哪些？
7．如何提高软件的可维护性？
8．什么是软件维护的最新方法？

第10章 软件管理

10.1 软件过程改进模型 CMMI

10.1.1 软件过程能力

过程(Process)是用于生产以及软件进化的一系列的活动、方法及实践。软件过程(Software Process)是将用户的需求转化为有效的软件解决方案的一系列活动的集合,包括定义的一系列软件开发中采用的规则和方法、人员的组织和培训以及开发的工具和设备。软件过程把这些要素科学地定义并合理地组织起来,把软件技术、人员技能、管理水平和组织机制整合在一起,以完成给定的项目目标。

但是软件过程并不能保证软件是高质量的,因为软件过程不能保证软件产品能够按期交付,也不能保证软件产品能够满足用户的要求。自从软件工程概念被提出之后,软件生命周期模型就成为了指导软件项目开发过程的重要手段。然而,即使针对项目类型裁剪软件过程而形成适合于本项目的过程模型,软件项目仍然会出现进度延迟、成本超支、质量达不到要求的情况。其主要原因是:软件过程模型中定义的软件过程只是名义上的,过程的实际执行与管理等实践才是软件过程能力的保证措施。

软件过程能力(Software Process Capability)描述了软件开发团队遵循某个软件过程规范后能够达到的预期结果的界限范围。它是对能力的一种衡量,可以用来预测一个软件开发组织在承接下一个软件项目时,所能期望得到的最可能结果。在遵循某个软件过程规范后得到的实际结果称为软件过程性能(Software Process Performance)。与软件过程能力的区别在于,软件过程能力关注的是期望得到的结果,而软件过程性能关注的是实际得到的结果。由于项目要求和客观环境的差异,软件过程性能不可能充分反映软件过程整体能力,即软件过程能力受限于它的环境。因此,一个软件团队要达到软件过程能力,除了通过软件生命周期模型规划和定义软件过程之外,还需要遵循定义好的软件过程规范进行具体实施,在过程实施中对部属的过程进行度量,发现过程性能与过程能力偏差较大时,应该及时对过程进行调整,即进行过程控制,并将这些调整内容纳入到统一的过程管理规范中。

软件过程成熟度(Software Process Maturity)是指一个具体的软件过程被明确地定义、管理、评价、控制和产生实效的程度。所谓成熟度包含着能力的一种增长潜力,同时也表明了软件组织实施软件过程的实际水平。随着软件过程成熟度能力的不断提高,在软件组织内部,通过把这一过程规范化,并对组织成员进行培训,使得软件过程可以被很好地理解,而

且可以持续地被过程的执行者关注、修改和完善。从而使软件的质量、生产率和生命周期得到改善。

因此，一个软件项目开发队伍除了要对软件生命周期模型进行裁剪外，还需要在实际遵照过程规范执行的过程中对软件过程进行控制，根据过程执行状态不断提出对过程的改进措施，从而真正实现软件过程的能力。显然，软件过程能力的提高需要对当前的软件过程状况进行科学地评估，如何对软件过程的执行状态进行评估，如何提出过程的改进方向和路线，这些问题就是CMM(Capability Maturity Model For Software，SW-CMM，简称CMM)/CMMI(Capability Maturity Model Integration)所需要解决的。

10.1.2 软件能力成熟度模型CMM

1. CMM发展概况

20世纪70年代中期，软件管理工程引起了人们的广泛关注。当时美国国防部曾立项专门研究软件项目做不好的原因，发现70%是因为管理不善而引起，而并不是因为技术实力不够，进而得出一个结论，即管理是影响软件研发项目全局的因素，而技术只影响局部。到了20世纪90年代中期，软件管理工程管理不善的问题仍然存在，大约只有10%的项目能够在预定的费用和进度下交付。软件项目失败的主要原因有：需求定义不明确；缺乏一个好的软件开发过程；没有一个统一领导的产品研发小组；子合同管理不严格；没有经常注意改善软件过程；对软件构架很不重视；软件界面定义不善且缺乏合适的控制；软件升级暴露了硬件的缺点；关心创新而不关心费用和风险；标准太少且不够完善等。在关系到软件项目成功与否的众多因素中，软件度量、工作量估计、项目规划、进展控制、需求变化、风险管理等都是与工程管理直接相关的因素。由此可见，软件管理工程的意义至关重要。

为了保证软件产品的质量，20世纪80年代中期，美国联邦政府提出对软件承包商的软件开发能力进行评估的要求。而CMM模型正是基于多年软件产品质量研究成果所建立的。美国的Walter Shewart于20世纪30年代发表了统计质量控制成果。在Watts Hunaphrey、Ron Radice等人的研究成果之上，美国卡莱基·梅隆大学软件工程研究所(Carnegie Mellon University，CMU/Software Engineering Institute，SEI)于1987年研究并发布了软件过程成熟度框架，并提供了软件过程评估(Software Process Assessment，SPA)和软件能力评价(Software Capability Evaluation，SCE)两种评估方法，以及软件成熟度提向单评估工具。1990年SEI将软件过程成熟度框架进化为软件能力成熟度模型CMM并公布了CMM 0.0版。1991年SEI公布了包含第二级关键过程域(Key Process Area，KPA)方案的CMM 0.4版及包含第三级方案的CMM 0.5版，同年，又发布了包含第四级和第五级KPA方案的CMM 0.7版。1991年底CMM 1.0版发布，它把软件过程按完善程度分为五个等级，描述了不同完善程度软件过程的不同特点。这个方法本是美国军方委托研究，用来评估军用软件承包商的软件过程、评价其软件开发能力的。但在试用过程中，该方法的另一个更加重要的作用越来越被人们重视，那就是它描述了软件过程不断改进的科学途径，使软件开发组织能进行自我分析，找出尽快提高软件过程能力的策略。这个方法的意义得到了国际软件产业界和软件工程界的广泛关注和认可，人们认为这是20世纪90年代

代软件工程技术最重要的发展之一。

经过两年的试用,1993 年 SEI 正式发布了 CMM 1.1 版,这是目前使用最为广泛的版本。1995 年,个体软件过程(Personal Software Process,PSP)又被提出,用于控制和改进个人软件开发方式。PSP 是一个过程描述、检测和方法的集合,能够帮助软件工程师改善其个人软件开发性能。它与具体的技术(程序设计语言、工具或者设计方法)相对独立,其原则能够应用到几乎任何软件工程任务之中。PSP 能够说明个体软件过程的原则;帮助软件工程师预估和计划其工作;确定软件工程师为改善产品质量要采取的步骤;建立度量个体软件过程改善的基准;确定过程的改变对软件工程师能力的影响;减少软件工作缺陷并提高计划和生产效率。PSP 注重于个人的技能,能够指导软件工程师如何保证自己的工作质量,估计和规划自身的工作,度量和追踪个人的表现,管理自身的软件过程和产品质量。经过 PSP 学习和实践的正规训练,软件工程师们能够在其参与的项目工作之中充分利用 PSP,从而保证了项目整体的进度和质量。CMM 是适用于软件开发组织中的流程管理,而 PSP 则面向个体开发人员。

CMM 基于众多软件专家的实践经验,是对软件组织在定义、实施、度量、控制和改善其软件过程的实践中各个发展阶段的描述。它的核心是把软件开发视为一个过程,并根据这一原则对软件开发和维护进行过程监控和研究,以使其更加科学化、标准化,使企业能够更好地实现商业目标。CMM 主要用于软件开发过程和能力的评价和改进,侧重于软件开发过程的管理及工程能力的提高与评估,是软件开发组织进行软件过程改进与评估的一个有效指导框架。自 20 世纪 90 年代以来,CMM 在北美、欧洲和日本成功地应用,现已成为事实上的软件过程改进工业标准,是软件业最权威的评估认证体系。

CMM 的出现是为了克服软件生产的危机。所谓软件生产的危机是指尽管新的软件开发方法和技术不断出现,但软件生产率和质量并未得到有效提高,软件产品不能按时完成,软件生产预算超支,而且交付客户使用的软件产品(特别是大型软件工程)中由于各种原因产生的错误无法克服。在 20 世纪 80 年代末期,美国国防部门和工业界开始认识到在软件开发中最重要的问题在于软件生产商对软件的生产过程管理不力,也就是说,软件生产过程的成败比新技术和开发方法更能决定一个项目或企业的成败。没有完善的软件生产过程体系,软件开发的成败只能依靠人为主观或偶然因素——比如某一杰出软件天才或小组的成就,而非可持续的客观标准及体系。因此,对成功软件过程的重复使用,对以往经验或教训的分析总结,对全部开发案例的系统编档存档就成了一套完整而成熟的软件过程。这是一个从无序到有序,从人为到客观标准,从定性到定量的不断积累与完善的过程。在该过程的演变中,软件组织会面临一系列有代表意义的成熟阶段。由此,美国 SEI 提出了软件能力的评价与改进指导体系。

CMM 的基本思想是:因为问题是由管理软件过程的方法使用不当引起的,所以新软件技术的运用并不会自动提高生产率和软件质量。能力成熟度模型有助于软件开发组织建立一个有规律的、成熟的软件过程。改进后的过程将开发出质量更好的软件,使更多的软件项目避免时间延误、费用超支等问题。软件开发组织可以依据 CMM 的框架对项目管理和项目工程进行定量控制和能力评估,而软件应用单位也可依据 CMM 来衡量和预测项目承接方的实际软件生产能力。这样,软件开发方与产品用户方都基于一个同样的标准来对软件生产和管理进行评测与控制。

2. CMM 结构

CMM 将软件组织的管理水平划分为五个级别：初始级（CMML1）、可重复级（CMML2）、定义级（CMML3）、定量管理级（CMML4）和优先级（CMML5），共计 18 个关键过程域，52 个具体目标，316 个关键实践。为了在软件过程改进中真正体现 CMM 的可操作性，CMM 按照层次给出了每一个成熟度等级的详细结构，如图 10-1 所示。每一个成熟度等级从其内部结构上可细分成许多组成部分，除第一级外，每一个成熟度等级的结构可以被自顶向下进一步被划分描述。

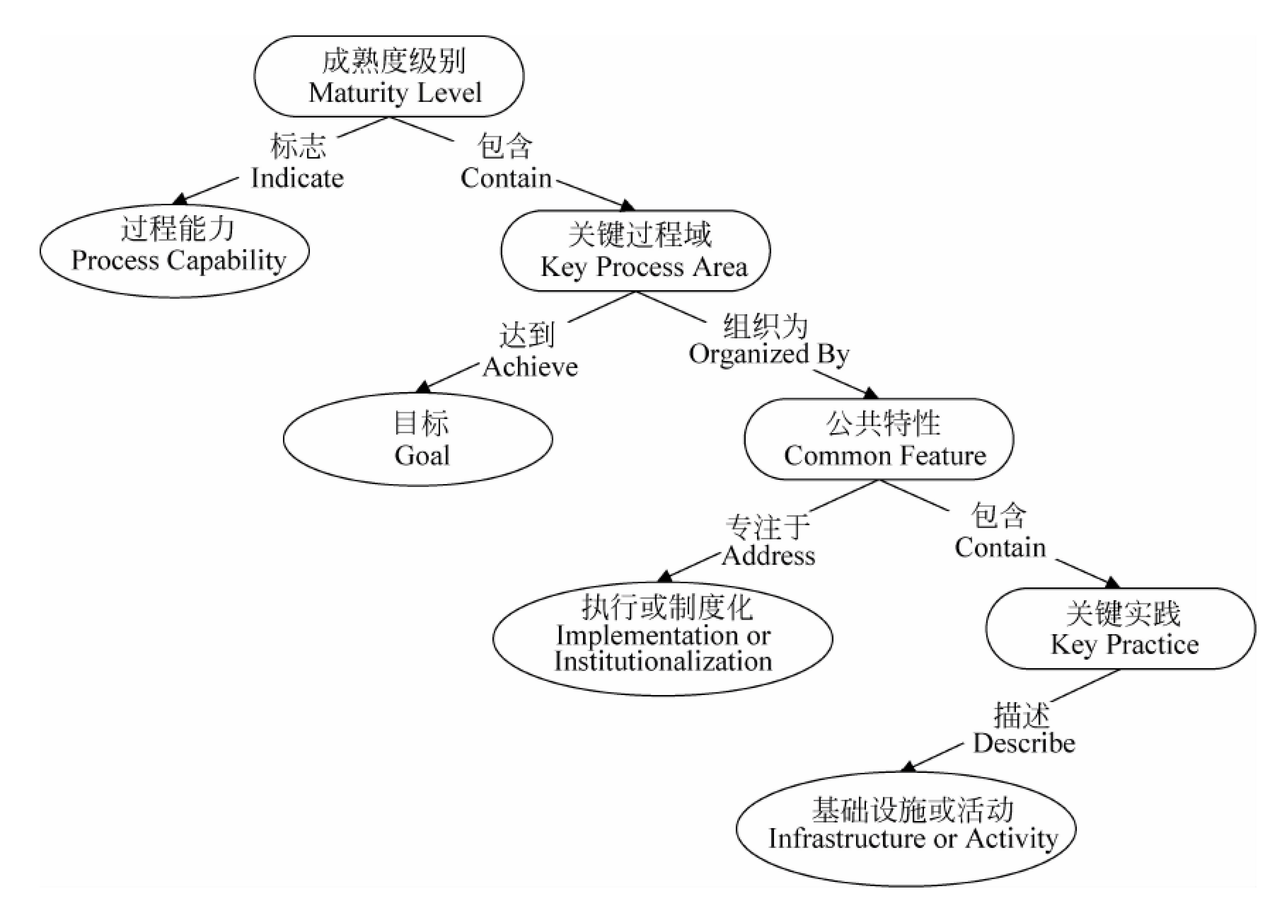


图 10-1　CMM 的层次结构

1）成熟度等级（Maturity Level）

经过深入地调查研究，软件开发人员认识到对软件过程的改进不可能一朝一夕就能成功，需要不断地进行软件过程改进工作，在完成一个又一个小的改进步骤基础上持续发展，而不是一蹴而就地彻底革命。CMM 为了较全面地描述和分析软件过程能力的发展程度，建立了一个软件过程成熟程度的分级标准，把软件过程从无序到有序的进化过程分成五个阶段，并把这些阶段排序，形成五个逐层提高的等级。这五个成熟度等级定义了一个有序的尺度，使软件组织可以评估其当前的过程成熟程度，并通过提出更严格的软件质量标准和过程改进，来选择进一步的改进策略，以达到更高级别的成熟度。

从 CMML1 至 CMML5，CMM 对每个成熟度级别特性的描述，说明了不同级别之间软件过程的主要变化。反映出一个软件组织为了从一个无序的、混乱的软件过程进化到一种有序的、有纪律的且成熟的软件过程，所必须经历的改进途径。每一个成熟度级别都是该软件组织改进途径中的一个台阶，后一个成熟度级别是前一个级别的软件过程进化目标。CMM 的每个成熟度级别中都包含了一组过程改进的目标，满足这些目标后，一个组织的软

件过程就从当前级别进化到下一个成熟度级别。相应地，该组织的软件过程都将得到一定程度的完善和优化，也使得过程能力得到提高。随着成熟度级别的不断提高，该软件组织的过程改进活动会取得更加显著的成效，从而使软件过程得到进一步地完善和优化。CMM就是以上述方式支持软件组织改进其软件过程活动的。

CMM 的五个成熟度等级如图 10-2 所示。

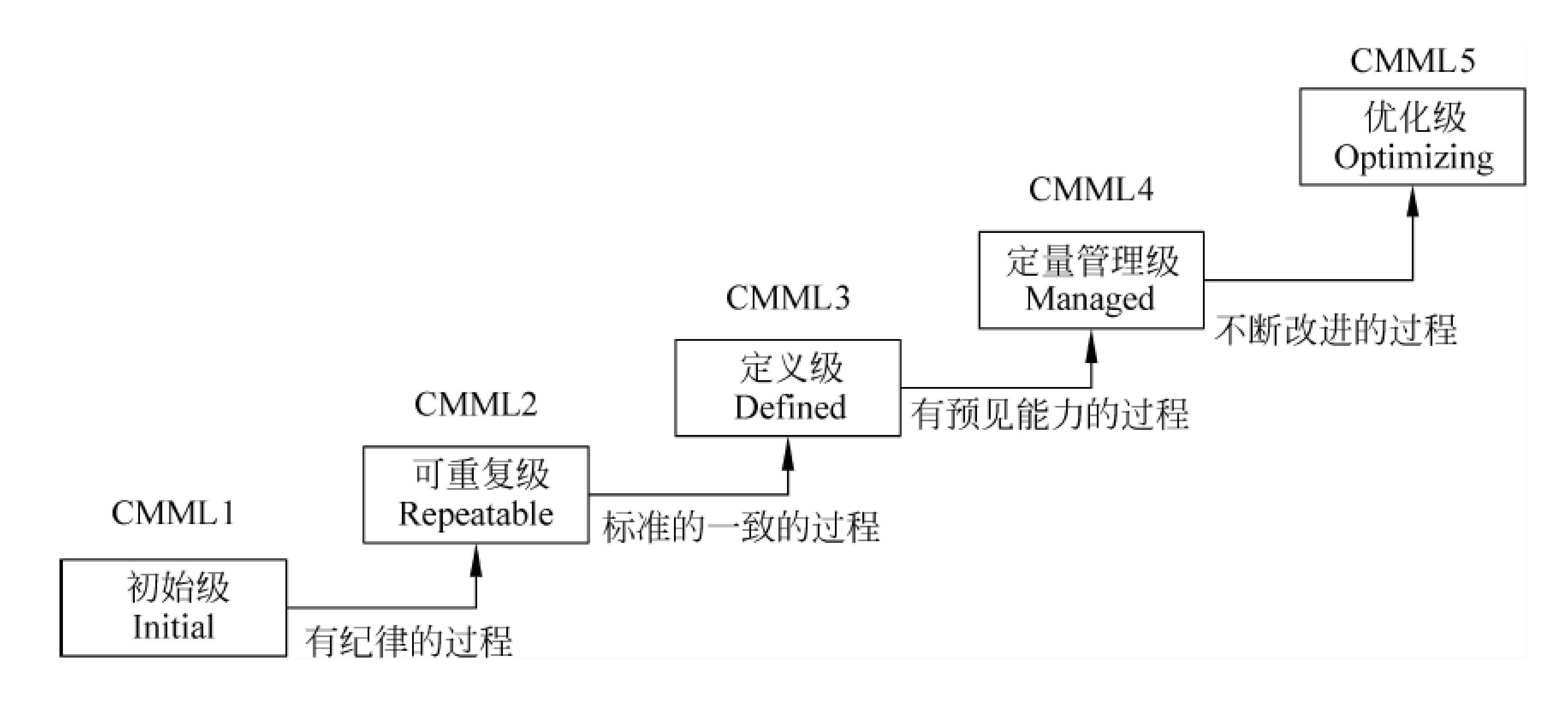


图 10-2 CMM 的成熟度等级

CMML1：初始级

在初始级，软件组织一般不具备稳定的软件开发与维护的环境，常常在遇到问题的时候放弃原定的计划而只专注于编程与测试。初始级软件开发组织一般具有以下特征。

- 软件过程的特点是杂乱无章，有时甚至是混乱，几乎没有定义过程的规则或步骤。
- 做出过分的承诺，或高目标的许诺，但实际上却出现一系列的问题。
- 遇到危机就放弃原计划过程，反复编码和测试。
- 成功完全依赖于个人和杰出的专业人才。具体的表现和成果都源于个人的能力和他们先前的经验、知识、进取心和积极程度。
- 能力只是个人的特性，而不是开发组织的特性。此类人一旦离去，对组织的稳定作用也就消失。
- 软件过程是不可确定的和不可预见的。软件成熟度处于第一级软件组织的软件过程，在实际的工作中被经常改变(过程是随意的)。也就是说，软件的计划、预算、功能和产品的质量都是不可确定的和不可预见的。

在初始级软件组织的开发过程中，能力只是个人行为不是组织行为，一旦人员流动或变更，整个组织的开发能力也随之改变。整个组织没有稳定的过程规则可依据，现有的种种规章制度也互不协调或相互矛盾；开发人员的工作方式是救火式的，哪里有漏洞就填补哪里，很少收集关于开发过程的数据；新技术的引进也要冒极大风险。总之，整个软件生产过程是不可重复的、不可预见的、不成体系的、不可积累的及不稳定的。

初始级的主要改进措施包括：

- 建立软件项目开发过程，实施规范化管理，保障项目的承诺。
- 进行需求管理，建立客户与软件开发组织之间的共同理解，使项目真正反映客户的要求。

- 建立完善的文档体系，包括软件开发计划、软件质量保证计划、软件配置管理计划、软件测试计划、风险管理计划及过程改进计划。
- 严格执行质量监控。
- 按 CMML2 所规定的各项核心实践进行开发。

CMML2：可重复级

确定了基本的软件生产管理和控制，能针对特定软件项目制定开发过程及管理措施，能将以往项目开发经验用于类似的新项目，有不同的软件生产过程可供不同的项目选择。能够客观预测并有效追踪软件生产成本和工期，过程标准在项目实施中能保证被遵循。项目的开发是有计划、有控制，并可重复的行为，总原则是：一个可管理的过程是一个可重复的过程，并能逐渐改进和成熟。可重复级的管理过程包括需求管理、项目计划、项目追踪和监控、子合同管理、质量保证与配置管理六个方面，一般具有以下特征：

- 可以给用户较有保证的承诺，因为软件组织可在以往同类项目的成功经验上总结和建立起一整套过程准则来保证成功地重复。
- 项目管理采用基线(Baseline)来标识进展并对成本和进度进行追踪。
- 组织通过子合同管理与用户建立有效的供求关系。
- 面对开发缺陷，有规则可以依据来纠正错误。
- 个人行为被稀释并分解到组织整体的规则和管理框架之中。
- 文档的准备和项目数据的收集相应完备。

可重复级的主要改进措施包括：

- 将各项目的过程经验总结为整个软件开发组织的标准过程，使其过程能力得以提高。
- 注意跨项目间的过程管理的协调和支持。
- 树立全组织的软件过程标准概念。
- 建立软件工程过程小组，对各项目的过程和质量进行评估和监控，使软件过程得以正确地调整。
- 积累数据，建立软件工程数据库和文档库。
- 加强培训。

CMML3：定义级

在定义级，软件过程已被编制为各个标准化过程，并在软件开发组织范围内执行，从而使软件生产和管理更具可重复性、可控制性、稳定性和持续性。其主要特征有：

- 过程在整个软件开发组织范围内得以确立。
- 制定了一套软件过程规则对所有软件工程和管理行为给予指导。
- 软件组织有了标准化的过程，并且可在所开发的项目中依据具体项目的需要，将标准过程调整为合适的项目过程。
- 组织内部设置了软件工程小组负责过程的制定、修改、调整和监督。该小组直接向软件组织最高领导层汇报。
- 建立培训机构专门对全组织员工进行过程培训。
- 各项目组的开发经验可相互借鉴并支持，对项目成本、工期及质量均可进行最终控制。

- 有关软件工程及管理工程的过程文件被编制并成为组织标准，所有项目都必须按照这些标准过程或经调整后的项目过程来实施，从而保障了每一次工程开发的投入和时间、项目计划、产品功能及软件质量得以控制。
- 软件过程在此得到稳定、重复和持续性的应用，使开发风险大为下降。
- 在开发的人员组织上，以项目组的方式进行工作，如同综合产品团队。各项目组人员参与软件过程的制定和修改，并引进符合项目过程的新软件开发技术，在各项目开发过程中收集的数据被系统共享。
- 能有计划地按人员的角色进行培训。

定义级的主要改进措施包括：

- 应准备对整个软件过程，包括生产和管理两方面的定量评测分析，以便尽可能将软件工程所涉及的定性因素转变为定量标准，从而对软件进行定量控制和预测。
- 应使整个组织的软件能力在定量基础上可预测和可控制。

CMML4：定量管理级

第四级的过程是量化过程。软件组织对产品与过程建立起定量的质量目标，同时对过程中加入规定得很清楚的连续度量。作为组织的度量方案，要对所有项目的重要过程活动进行生产率和质量的度量，软件产品因此具有可预期的高质量。定量管理级一般具有以下特征：

- 所有项目和产品的质量都有明确的定量化衡量标准，软件也被置于这样一个度量体系中进行分析、比较和监控。所有定量指标都被尽可能地详细采集并描述，使之可具体应用于软件产品的控制之中。
- 软件开发真正成为一种工业化生产行为，由专门的软件过程数据库收集和分析软件过程中的各类数据，并以此作为对软件活动质量评估的基准。
- 软件组织所有项目的生产过程在量化的基础上大大提高了可控制性和可预测性，生产过程中可能面对的偏差被控制在一定的量化范围内并被分析和解决；新技术的采纳也在量化基础上有控制地进行，从而减小了风险。
- 所有的软件过程和产品都树立了定量的目标并被定量地管理，从而可以很好地预测软件组织的能力。
- 此阶段中所有定量标准都是明确定义并持续一致的，可以用于对软件过程和管理的评估与调节。
- 所有修正和调节方法(包括对偏差及缺陷的校正分析)都是基于变化指标上的；新的软件开发技术也在定量的基础上被评估。
- 项目组成员对整个过程及其管理体系有高度一致的理解，并已学会运用数据库等方法定量地看待和理解软件工程。

定量管理级的主要特点是定量化，可预测化和高质量，其主要改进措施包括：

- 注意采取必要措施与方案以减少项目缺陷，尽量建立起缺陷防范的有效机制。
- 引进技术变动管理以发挥新技术的功能。
- 引进自动化工具以减少软件工程中的人为误差。

- 实行过程管理，不断改进已有的过程体系。

CMML5：优化级

在这个等级，整个软件组织将会把重点放在对过程进行不断地优化，主动找出过程的弱点与长处，以达到预防缺陷的目标。同时，分析有关过程有效性的资料，做出对新技术成本与收益的分析，以及提出对过程进行修改的建议。其特征为：

- 软件过程是持续改进的过程，并且有一整套有效机制确保软件工程误差接近最小或零。
- 每一个过程在具体项目的运用中，可根据周边和反馈信息来判断下一步实施所需要的最佳过程，以持续改善过程使之最优化。因此，软件组织能不断调整软件生产过程，按优化方案改进并执行所需过程，把精力集中于持续的过程改进之中。
- 新技术的采用也被作为日常活动加以规划，各项目组已具备尽早和尽快识别工程缺陷并改正错误的手段。这需要完善的数据库和长期积累的量化指标来协助实现，新技术和自动化工具也使软件工程人员能够预防软件缺陷并找到其根源以防止错误再现，软件组织资源在第五级阶段被有效利用并节约。
- 组织在优化级所遵循的持续改进措施既包括对已有过程的渐进改善，也包括应用新技术和工具所产生的革新式改进。
- 在第四级的定量化标准基础之上，整个组织的过程定义、分析、校正和处理能力大大加强。
- 项目组都能主动找到产生软件问题的根源，也能对导致人力和时间浪费等低效率因素进行改进，防止浪费再次发生。
- 整个机构都有强烈的团队意识，每个人都致力于过程改进、缺陷防范和高品质的追求。

成熟度等级是在朝着实现成熟软件过程进化途中的一个妥善定义的平台。五个成熟度等级构成了CMM的顶层结构，每一个成熟度等级包含若干个关键过程域，标志着处于该等级的软件组织具有一定水平的软件过程能力。CMM通过定义能力成熟度的五个等级，引导软件开发机构不断识别出其软件中的缺陷，并指出应该做哪些改进，但是，它并不提供做这些改进的具体措施。

2）过程能力（Process Capability）

过程能力是描述软件过程的能力，预测通过遵循软件过程能实现预期结果的程度。一个组织的软件过程能力提供一种“预测该组织承担下一个软件项目时，最可能得到的结果”的方法。

3）关键过程域（Key Process Area，KPA）

KPA规定了软件组织为了达到某个成熟度等级而必须满足的条件，它也是软件组织向该成熟度等级迈进时必须集中力量改进的重要方面。实施每个KPA所包含的关键实践，就是实现此KPA所制定的目标并提高软件过程能力。

4）公共特性（Common Feature）

公共特性是一种属性，它能指示一个KPA的实施和规范化是否是有效的、可重复的和持久的，CMM把关键实践分别归入五个公共特性之中。

5）目标(Goal)

目标用于确定一个组织或项目是否已有效地实施了一个KPA，即目标是检验KPA是否满足的一个指标。目标确定每个KPA的范围、边界、意图和关键实践。例如，KPA“软件项目计划”的一个目标是“软件估算已经文档化，供计划和跟踪软件项目使用”。

6）关键实践(Key Practices)

每个KPA都用若干关键实践描述，实施关键实践有助于实现相关的KPA目标。关键实践描述了对KPA的有效实施和规范化贡献的最大基础实施和活动。例如，在关键过程域“软件项目计划”中，一个关键实践是“按照已文档化的规范制定项目的软件开发计划”。

各个KPA中的关键实践都可按公共特性进行归类，每个KPA都包括五类关键实践。

(1) 执行保证(Commitment To Perform)：为完成KPA中的目标组成所需的承诺称为执行保证，它是软件组织执行特定的KPA所拟定的指导开发过程的规则和项目管理责任。

(2) 执行能力(Ability To Perform)：指软件组织执行KPA必须达到的前提条件，包括资源、组织机构、人员培训等多种措施。对KPA的执行必须建立在此基础之上，才可保证所规划的目标得以实现。

(3) 实施活动(Activities Performed)：描述了执行KPA所需要采纳的必要行动和步骤，与项目执行息息相关，包括计划、跟踪、检测等，这是关键实践的五种分类中与项目执行唯一相关的属性，其余四个属性都关注于软件组织的基础能力建设。

(4) 度量与分析(Measurement And Analysis)：是关于过程的定量度量和分析，以确定所执行活动的效果并据此做出分析判断。

(5) 实施验证(Verify Implementation)：在过程执行的中途及末尾对过程实施进行验证以确保执行活动与制定的过程相一致。它包括检测、复审等一系列质量保证活动，这些质量保证活动需要通过项目组以外的独立质检人员和管理人员来保证验证的有效执行，从而确保软件产品符合计划要求。

应该指出，关键实践只是规定了软件过程必须达到什么样的标准而未规定这些标准应如何实现，因此，对同样的过程水平，不同软件组织，不同项目可采纳不同的过程和实施方式去完成。

3. CMM应用

CMM有两个基本用途：软件过程评估和软件能力评价。

(1) 软件过程评估：目的是确定一个组织的当前软件过程的状态，找出组织所面临的急需解决的与软件过程有关的问题，进而有步骤地实施软件过程改进，使组织的软件过程能力不断提高。

(2) 软件能力评价：目的是识别合格的、能完成软件工程项目的承制方，或者监控承制方现有软件工作中软件过程的状态，进而提出承制方应改进之处。

由于软件过程评估和软件能力评价是有着不同目的的两种应用，因此所用的具体方法有明显差异。但是两者都以CMM模型及其衍生产品为基础，实施的基本步骤也一致。

1）软件过程评估和软件能力评价的基本方法和步骤

软件过程评估关注一个组织的软件过程有哪些需要改进之处及其轻重缓急。评估组采用CMM来指导他们进行调查、分析和排优先次序。组织可利用这些调查结果，参照CMM

的关键实践所提供的指导，规划本组织软件过程的改进策略。

软件能力评价关注识别一个特别项目在进度要求和预算限制内构造出高质量软件所面临的风险。在采购过程中可以对投标者进行软件能力评价。评价的结果，可用于确定在挑选承制方方面的风险；也可对现有的合同进行评价以便监控承制方的过程实施，从而识别出承制方的软件过程中潜在的可改进之处。

CMM 为进行软件过程评估和软件能力评价建立一个共同的参考框架，作为评估软件成熟度的根据。图 10-3 概要地描述了评估和评价中的共同步骤。

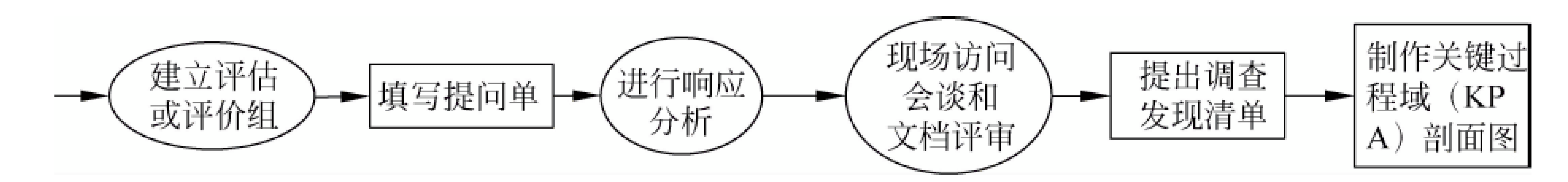


图 10-3　软件过程评估和软件能力评价的共同步骤

第一步：建立一个小组。该小组的成员应该是具有丰富软件工程和管理知识的专业人员。对该小组进行 CMM 基本改建和评估或评价方法细节方面的培训。

第二步：填写提问单。让待评估或评价单位的代表完成成熟度提问单的填写和其他诊断工具的要求。

第三步：进行响应分析。评估或评价组对提问单响应进行统计分析，定义必须做进一步调查的区域。待调查的区域与 CMM 的关键过程域相对应。

第四步：现场访问会谈和文档评审。访问被评估或评价单位的现场。评估和评价组根据响应分析的结果，召开座谈会，进行文档复审，以便了解该现场所遵循的软件过程。CMM 中的关键过程域和关键实践对评审或评价组成员在提问、倾听、复审和综合各种信息方面提供指导。确定现场关键过程域的实施是否满足相关的关键过程域的目标时，该组将运用专业性的判断。当 CMM 的关键实践与现场的关键实践间存在明显差异时，该组必须用文档记下对此关键过程域做出判断的理论依据。

第五步：提出调查发现清单。在现场工作阶段结束时，评估和评价组产生一个调查发现清单，明确指出该组织软件过程的强项和弱项。在软件过程评估中，该调查发现清单作为提出过程改进建议的基础；在软件能力评价中调查发现清单将作为软件采购单位所作风险分析的一部分。

第六步：制作关键过程域（KPA）剖面图。评估或评价组制作一份关键过程域剖面图，指出该组织已满足和尚未满足关键过程域目标的区域。一个关键过程域可能是已满足要求的，但仍存在一些相关的问题，如果未发现或未指出这些问题，就会妨碍实现该关键过程域的某个目标。

总之，软件过程评估和软件能力评价方法两者均：

- 采用成熟度提问单作为现场访问的出发点。
- 采用 CMM 作为指导现场调查研究的导引图。
- 采用 CMM 中的关键过程域生成明确地指出软件过程强项和弱项的调查发现清单。
- 在对关键过程域目标满足情况进行分析的基础上，衍生出一个剖面。
- 根据调查发现清单和关键过程域剖面，向合适的对象提出结论意见。

2）软件过程评估和软件能力评价之间的差异

尽管软件过程评估和软件能力评价有上述相似之处，但由于在动机、目的、输出和结果的所有权等方面均不同，导致两者在会谈目的、询问的范围、所采集的信息和结果的表达方式上有所不同，所采用的详细规程有别，培训要求也不一样。

- 软件过程评估是在开放的、合作的环境中进行的，评估目的在于暴露问题和帮助经理和工程师们改进软件过程的支持，一般都能得到较好的支持。评估的成功取决于管理者和专业人员对改进软件过程的支持，一般较容易取得成功。评估过程中虽然提问单是个重要工具，但更重要的是通过各种会谈了解组织的软件过程。评估的结果除了识别组织所面临的软件过程问题外，最有价值的还是，明确软件过程的改进途径，促进制订进一步的行动计划，使全组织关注改进过程，提高执行改进行动计划的动力和热情。
- 而软件能力评价是在更像审计和环境中进行。评价的目的与金钱密切相关，因为评价组的推荐性意见将影响挑选承制方或设置资金。评价过程的重点放在复审已文档化的审计记录上，这些记录能揭示组织实际执行的软件过程。

3）其他应用

除了上述两个基本用途外，CMM 在过程改进方面还有一些其他用途，主要是组织内负责软件过程改进的机构。例如，软件工程过程组(SEPG)，在策划改进措施、实施措施计划和定义过程中可以充分利用 CMM。

- 在策划改进措施期间，软件工程过程组可将 CMM 中关键过程域的目标与组织的当前实践相比较，仔细分析公司目标、管理优先级、实践运行的层次、实施每项实践对组织的价值以及组织在其文化背景下实施某项实践的能力等方面有关的关键实践。
- 软件工程过程组必须确定需要做哪些过程改进，如何实现变更以及如何获得所需要的支持。CMM 可以给有关过程改进的讨论提供一些初步的议题，帮助揭示与通用软件工程实践完全不同的前提条件。
- 在实施措施计划时，软件工程过程组可用 CMM 和关键实践来构造部分可操作的措施计划和定义软件过程。

随着企业 CMM 成熟度等级的提高，项目开发中的风险可以得到逐步减低，开发时间也会大大缩短，开发成本得以减少并可大大降低软件产品中的错误发生率。CMM 不仅可以提高企业在国际市场上的软件出口竞争力，也可提高企业自身的软件管理与开发水平，有助于客户对企业生产能力树立信心。目前，欧美等国的大型软件用户与软件供应商共同采纳 CMM 作为供需双方软件产品质量及工程预算的标准。印度软件企业更是对 CMM 全力投入，每年定期进行 CMM 培训。

10.1.3 软件能力成熟度模型集成 CMMI

按照 SEI 最初的计划，应该在 1998 年发表 CMM 2.0 版，由于软件过程评估(SPA)国际标准项目的进展，美国国防部下令暂时停止推进到 CMM 2.0 版，以便吸收 SPA 的长处，于是便产生了 CMMI，即软件能力成熟度模型集成。

自从 CMM 1.0 版发布以来，SEI 针对不同领域的要求对 CMM 先后进行改进，并衍生出了一系列成熟度模型，包括系统工程能力成熟度模型(SE-CMM，System Engineering

CMM)、软件采购能力成熟度模型(SA-CMM,Software Acquisition CMM)、集成产品开发能力成熟度模型(IPD-CMM,Integrated Product Development CMM)、人力资源能力成熟度模型(P-CMM,People CMM)等。CMMI 将包括上述能力成熟度模型在内的各种能力成熟度模型、继承产品和过程开发等整合到同一架构中去,由此建立起包括软件工程、软件采购和系统工程在内的诸模型集成,以满足除软件开发以外的软件系统工程和软件采购工作中的迫切需求。2000—2001 年 SEI 发布了系统工程和软件工程综合能力成熟度模型(CMMI-SE/SW,CMMI for System Engineering and Software Engineering)1. 0 版和 CMMI-SE/SW 1. 1 版。CMMI 兼收了 CMM 2. 0 版 C 稿草案和 SPA 中更合理、更科学和更周密的优点,是一套融合多学科的、可扩充的产品集合,同时也是工程实践与管理方法。它能够解决现有的、不同 CMM 的重复性、复杂性,并减少由此引起的成本,缩短改进工程。

CMMI 的源模型有三个：CMM 2. 0 版 C 稿草案；电子行业协会临时标准 731 (Electronic Industries Alliance Interim Standard 731)；IPD-CMM V0. 98 版。此外,SEI 在开发 CMMI 时还注意到了与国际标准化组织和国际电工委员会的 15504 技术报告(ISO/IEC 15504)相兼容和一致。与原有的 CMM 相比,CMMI 涉及面更广,专业领域覆盖软件工程、系统工程、集成产品、过程开发和系统采购,其表达方式有阶段式和连续式两种。这两种方式只是从不同的角度来阐述 CMMI,其实质上表达的内容是一致的。

1. 阶段式 CMMI

阶段式 CMMI 类似于 SW-CMM,将所有的过程域按五个成熟度等级来组织,称做“CMMI 成熟度等级”,其层次结构如图 10-4 所示。从低到高依次为：初始级(Initial)、已管理级(Managed)、已定义级(Defined)、量化管理级(Quantitatively Managed)和优化级(Optimizing)。每个成熟度等级都有一组 KPA 指出一个组织应集中于何处以改善其组织过程,每个 KPA 都用满足其目标的方法来描述,过程改进通过在一个特定的成熟度等级中满足所有 KPA 的目标而实现。

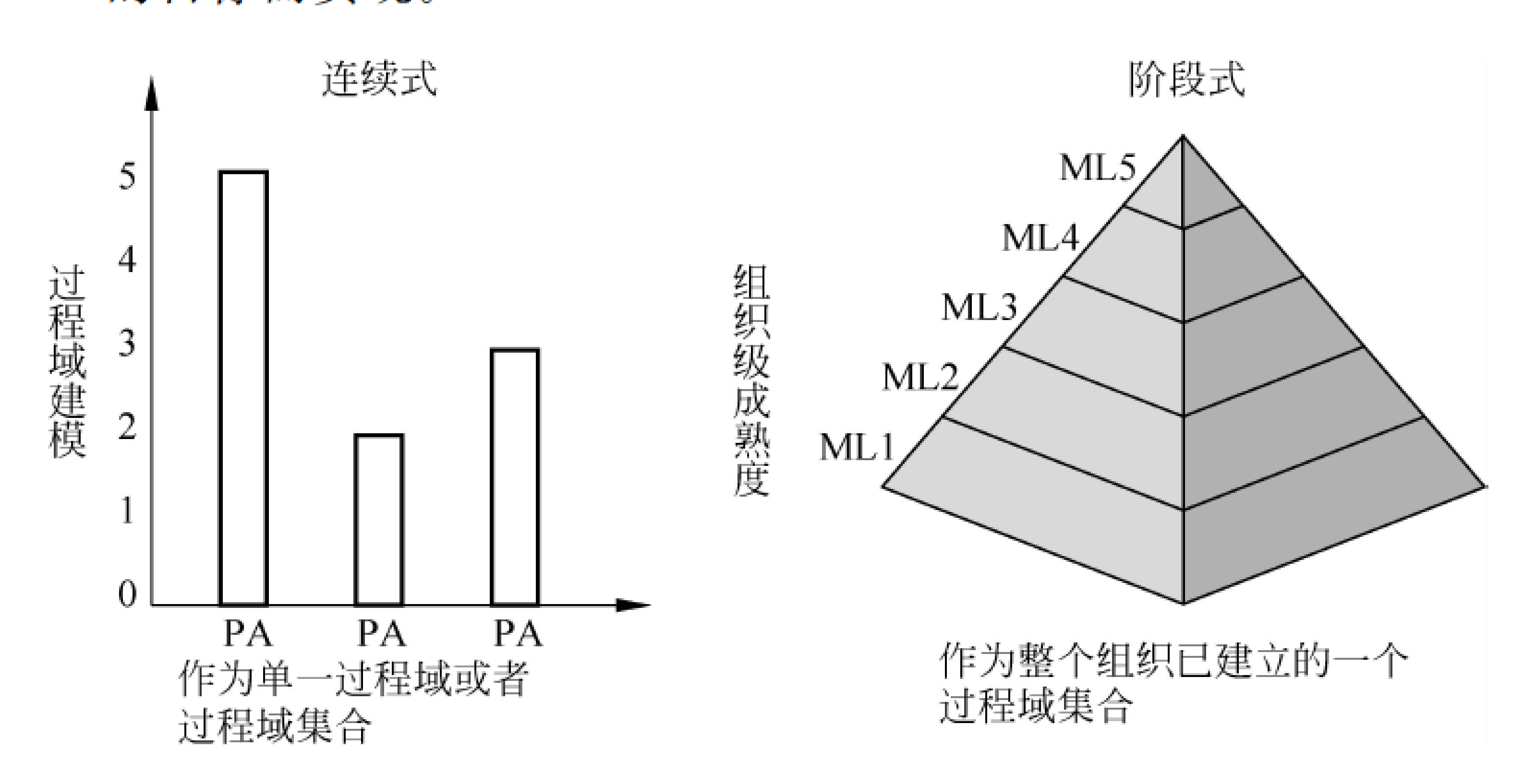


图 10-4　CMMI 模型类型

1) CMMI Level 1：初始级

软件开发组织对项目的目标与要做的努力很清晰,项目的目标得以实现,但在这一级上

的项目实施对实施人员有很大的依赖性。但是由于任务的完成带有很大的偶然性,组织无法保证在实施同类项目的时候仍然能够完成任务。

2)CMMI Level 2:已管理级

在项目实施上能够遵守既定的计划与流程,有资源准备,权责到人,对相关的项目实施人员有相应的培训,对整个流程有监测与控制,并与上级单位对项目与流程进行审查。软件组织在二级水平上体现了对项目的一系列的管理程序。这一系列的管理手段排除了在一级时完成任务的随机性,保证了软件组织的所有项目实施都会得到成功。

3)CMMI Level 3:已定义级

软件组织不仅能够对项目的实施有一整套的管理措施,保障项目的完成;而且,能够根据自身的特殊情况以及自己的标准流程,将这套管理体系与流程予以制度化,这样企业不仅能够在同类的项目上成功地实施,在不同类的项目上一样能够得到成功地实施。科学的管理成为软件组织的一种文化。

4)CMMI Level 4:量化管理级

项目管理不仅形成了一种制度,而且要实现数字化的管理,即对管理流程要做到量化与数字化。通过量化技术来实现流程的稳定性、实现管理的精度、降低项目实施在质量上的波动。

5)CMMI Level 5:优化级

项目管理达到了最高的境界,软件组织不仅能够通过信息手段与数字化手段来实现对项目的管理,而且能够充分利用信息资料,对软件组织在项目实施的过程中可能出现的次品予以预防。能够主动地改善流程,运用新技术,实现流程的优化。

阶段式的CMMI模型结构图,如图10-5所示。

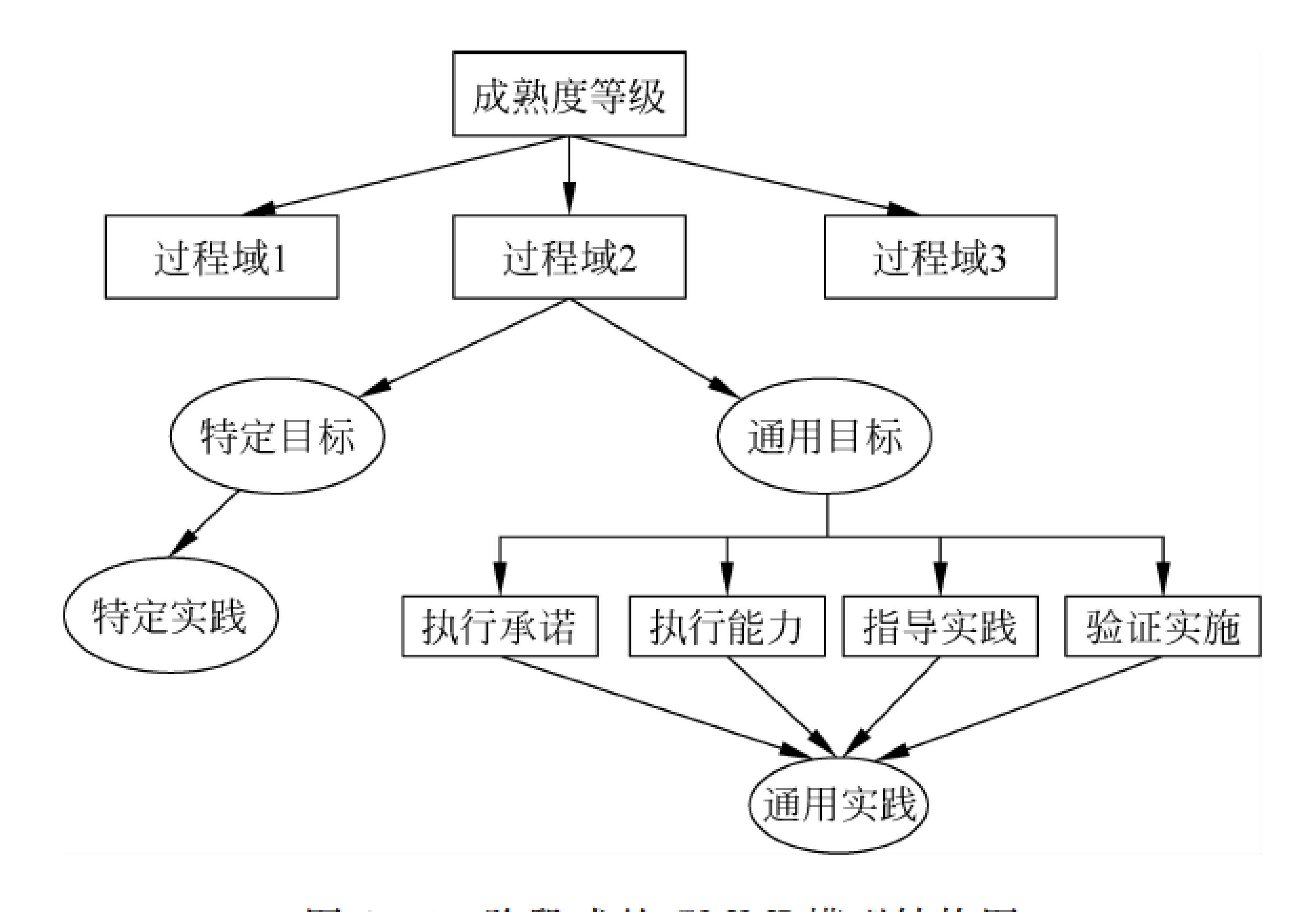


图10-5 阶段式的CMMI模型结构图

每一个成熟度等级包含了若干个过程域(Process Area,PA)。PA是为了实现若干给定的目标而要共同加以实施的一组实践,它是建立组织的过程能力所使用的主要构件,按照五个成熟度等级分组,每个PA的特定目标由其特定实践描述。

1）已管理级 PA

- 需求管理(Requirements Management)：维护项目的产品需求或部件需求，保持项目的计划、活动和工作产品与需求一致。
- 项目规划(Project Planning)：建立和维护规定活动的项目计划。
- 项目监督和控制(Project Monitoring and Control)：提供对项目过程足够的可视性，以便当项目性能显著偏离计划时能采取适当的纠正措施。
- 供应商协议管理(Supplier Agreement Management)：对从项目外部来源获得的产品和服务进行管理。
- 度量和分析(Measurement and Analysis)：开发并保持度量能力，以支持管理部门的信息需要。
- 过程和产品质量保证(Process and Product Quality Assurance)：通过对活动和工作产品遵守合适的要求、过程说明、标准和规程的情况进行客观评审，以提供对活动和工作产品的可视性。
- 配置管理(Configuration Management)：利用配置标识、配置控制、配置状态报告和配置审计，建立并维护工作产品的完整性。

2）已定义级 PA

- 需求开发(Requirements Development)：产生并分析客户、产品、产品部件的需求。
- 技术解决方案(Technical Solution)：设计、开发和实施满足需求的解决方案。
- 产品集成(Product Integration)：将产品部件组装成满足客户要求的软件产品。
- 验证(Verificaticra)：验证用户需求的完备性和正确性，验证开发过程中后一活动对前一活动要求的满足程度。
- 确认(Validation)：证实产品或者产品部件在期望的工作环境中能够实现用户期望的功能。
- 组织过程焦点(Organizational Process Focus)：在对组织过程及过程资产(Process Assets)的强弱进行彻底理解的基础上，规划并实施组织过程改进。
- 组织过程定义(Organizational Process Definition)：建立并维护可以使用的组织过程资产集合。
- 组织培训(Organizational Training)：开发员工的技能和知识，以便他们能有效地履行其职责，并产生较好的效果。
- 集成项目管理(Integrated Project Management)：从组织的标准过程集合中裁剪出集成的、已定义的过程，并以此过程来管理项目及相关内容。
- 风险管理(Risk Management)：预先识别潜在问题，使风险处理活动可以有计划，并在需要时用来减少对实现目标的负面影响。
- 决策分析和解决方案(Decision Analysis and Resolution)：对满足目标有显著影响的问题确定若干备选方案，分析备选方案，选择一种或多种能支持预定目标的最佳方案。

3）量化管理级 PA

- 组织过程性能(Oganizational Process Performance)：建立和维护对组织标准过程集合的定量理解，包括反映过程性能的数据、基线、模型，以便定量管理项目。

- 定量项目管理(Quantitative Project Management):定量地管理部署在项目中的已定义过程,以达到已建立的质量和过程性能目标。

4) 优化级 PA

- 组织创新和部署(Organizational Innovation and Deployment):为了支持组织质量和过程性能目标,选择并部署增量或者创新过程改进的过程和技术。
- 原因分析和解决方案(Causal Analysis and Resolution):识别缺陷和其他问题,分析产生的原因,采取措施预防它们再次发生。

CMMI 内容分为三个等级,必需的(Required)、期望的(Expected)和提供信息的(Informative),来衡量模型包括的质量重要性和作用。最重要的是"必需的"级别,是模型和过程改进的基础。第二级别"期望的"在过程改进中起到主要作用,但是某些情况不是"必需的"可能不会出现在成功的组织模型中。"提供信息的"构成了模型的主要部分,为过程改进提供了有用的指导,在许多情况下对必需的和期望的构件做了进一步说明。"必需的"模型构件是目标,代表了过程改进想要达到的最终状态,它的实现表示了项目和过程控制已经达到了某种水平。因此,PA 的目标可分为两种:一种是特定目标;另一种是通用目标。

特定目标是专门针对某个 PA 的唯一特征而制定的,描述为了实现该 PA 必须做的内容。特定目标是必需的模型部件。特定实践是为了达到特定目标而部署的重要活动,它是期望的模型部件。和特定目标只适用于某一个 PA 不同,通用目标适用于多个 PA,在阶段式 CMMI 中,每个 PA 只有一个通用目标。通用目标是必需的模型部件,实现通用目标表明:该 PA 相关过程的规划和实施是受控的,因此,这些过程是有效果的、可重复的和持续的。

类似于 CMM 中的公共特性(Common Feature),CMMI 中针对通用目标而部署的通用实践也被抽象为公共特性,但只有四类。

- 执行承诺(Commitment to Perform):包括确保过程能够建立并持续进行的实践,一般涉及组织的方针政策和领导责任。
- 执行能力(Ability to Perform):包括为彻底实施过程而建立必要资源条件的实践,一般涉及资源、组织结构、培训等。
- 指导实施(Directing Implementation):包指管理过程性能、管理工作产品完整性的实践,一般涉及过程和产品的度量数据。
- 验证实施(Verifying Implementation):包括高层管理人员的评审,以及过程目标评价等实践,以判断是否符合过程描述、步骤、标准等。一般涉及评审和审计。

CMMI 阶段表示法如图 10-6 所示。

2. 连续式 CMMI

连续式 CMMI 的层次结构如图 10-7 所示。

连续式 CMMI 将 PA 分为四类:过程管理、项目管理、工程以及支持功能,如图 10-8 所示。每个 PA 划分为六个能力等级,称做"CMMI 能力等级"。连续式 CMMI 的能力等级不再针对组织,而是针对 PA,如表 10-1 所示。

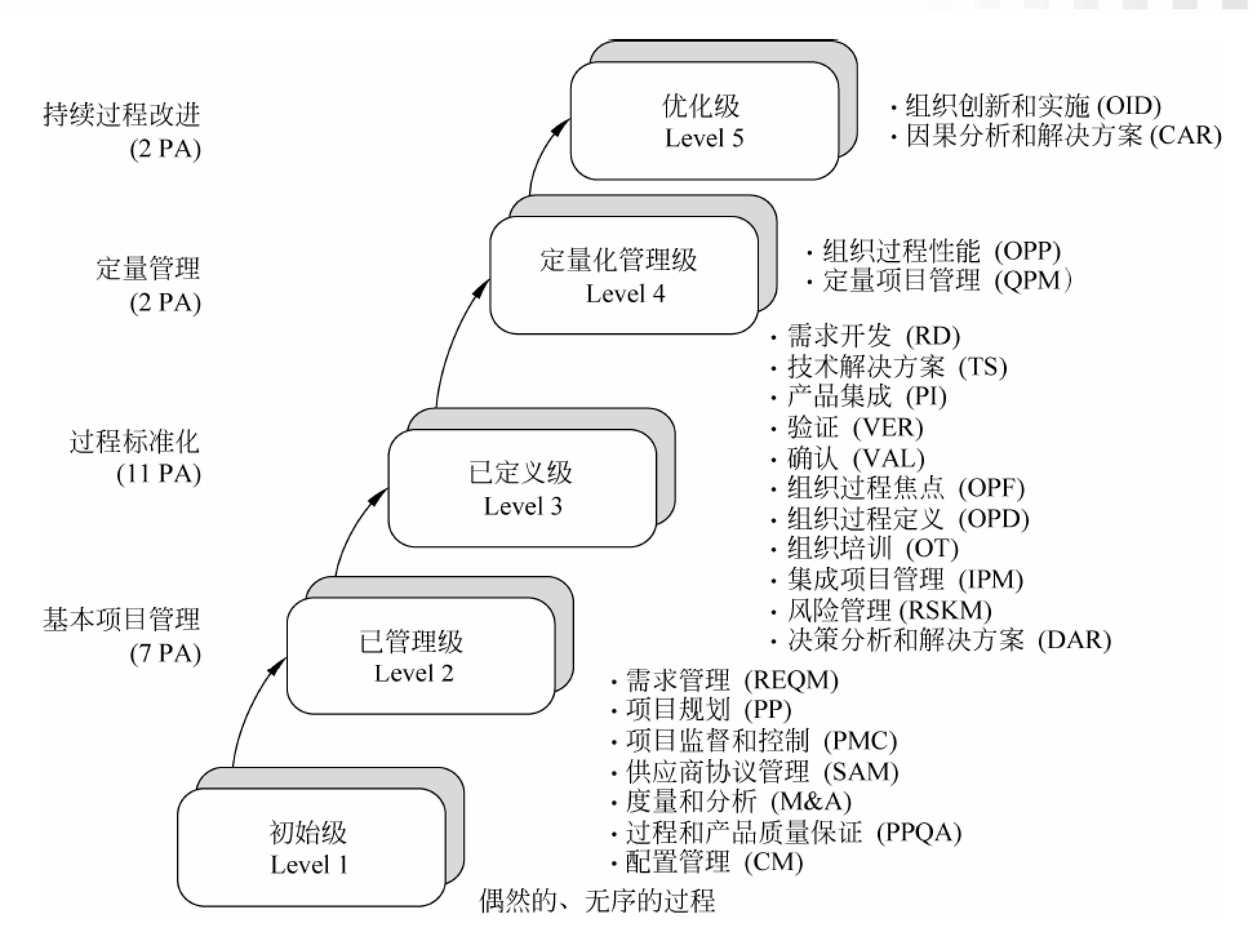


图 10-6 CMMI 阶段表示法

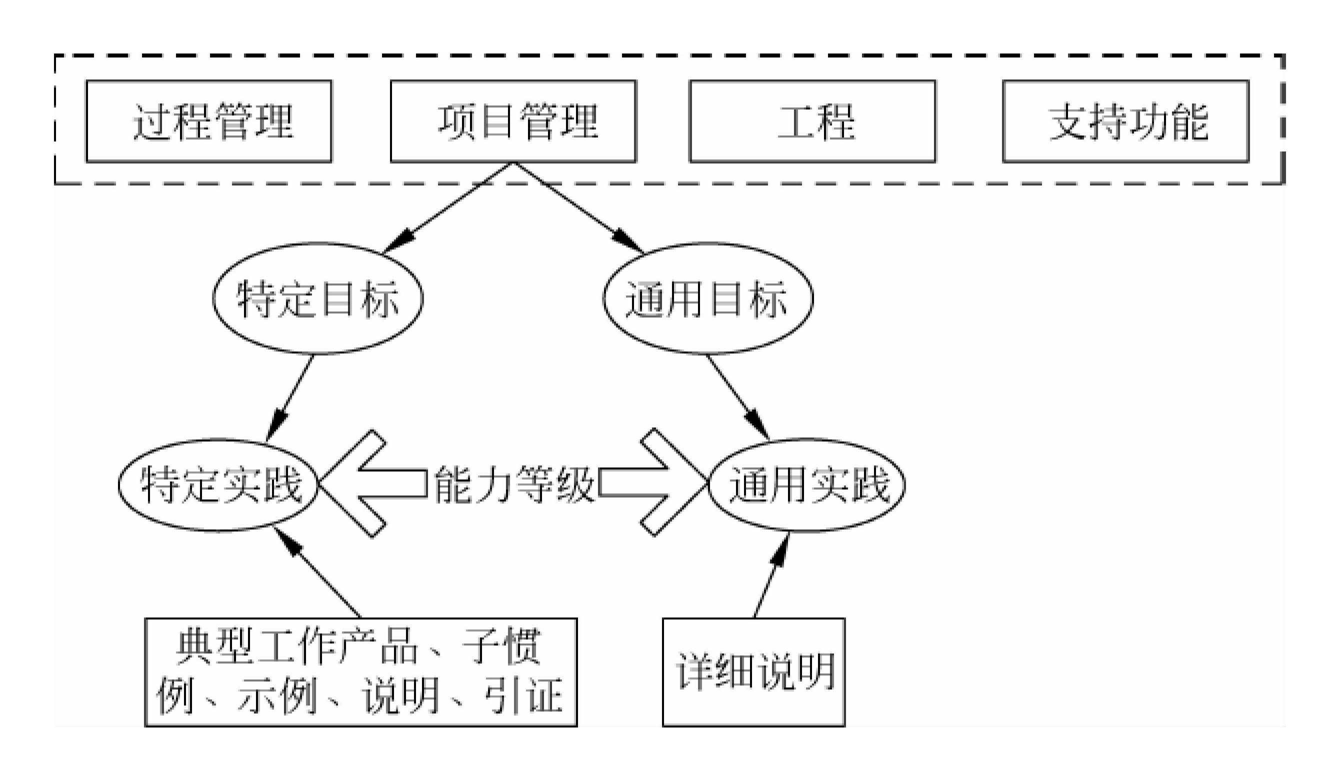


图 10-7 连续式 CMMI 层次结构

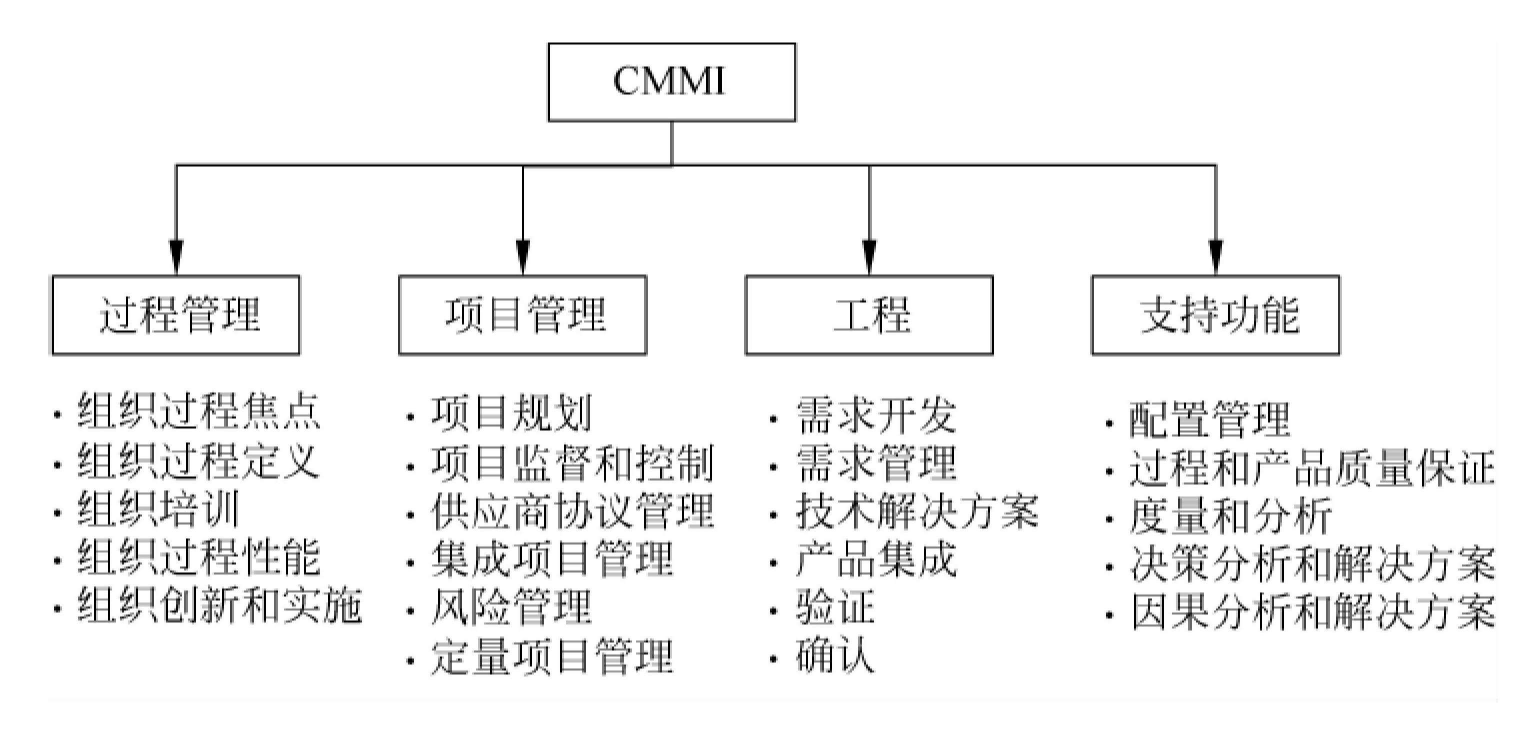


图 10-8 CMMI 连续表示法

表 10-1 连续式的能力等级

等级编号	等　　级	说　　明
0	未执行、未完成	
1	已执行	对某个过程域，特定实践（必要活动）被执行，基本满足了特定目标（此过程域的基本目的）
2	已管理	按照项目方式对"特定实践"组成的过程进行管理
3	已定义	按照组织统一的过程执行"特定实践"，并进行管理
4	定量管理	在过程域中引入度量，并利用度量进行管理
5	优化的	不断改进这个过程域的过程性能

连续式 CMMI 中没有专门陈述目标，而是更加强调实践。它在完成改善的次序上缺乏专门的指导，其实践以支持单个 PA 的改善和增长的方式来组织。这样，在按照连续式表示方法实施 CMMI 的时候，一个组织可以把项目管理或者其他某类的实践一直做到最好，而可以完全不必考虑其他方面的 PA。软件组织根据连续式评估的结果是一张"能力特征图"，用以描述组织各个 PA 的能力等级。组织可以通过定义各个 PA 的能力等级来确定改进的目标，即确定一张"能力特征图"为改进目标。

3. CMMI 实施方法 IDEAL

IDEAL 模型是一个软件组织用于启动、规划和实现软件过程改进措施蓝图的模型，概括了建立一个成功的过程改进项目的必要步骤，其中 I 代表 Initiating（启动）、D 代表 Diagnosing（诊断）、E 代表 Establishing（建立）、A 代表 Acting（实施）、L 代表 Learning（学习），如图 10-9 所示。

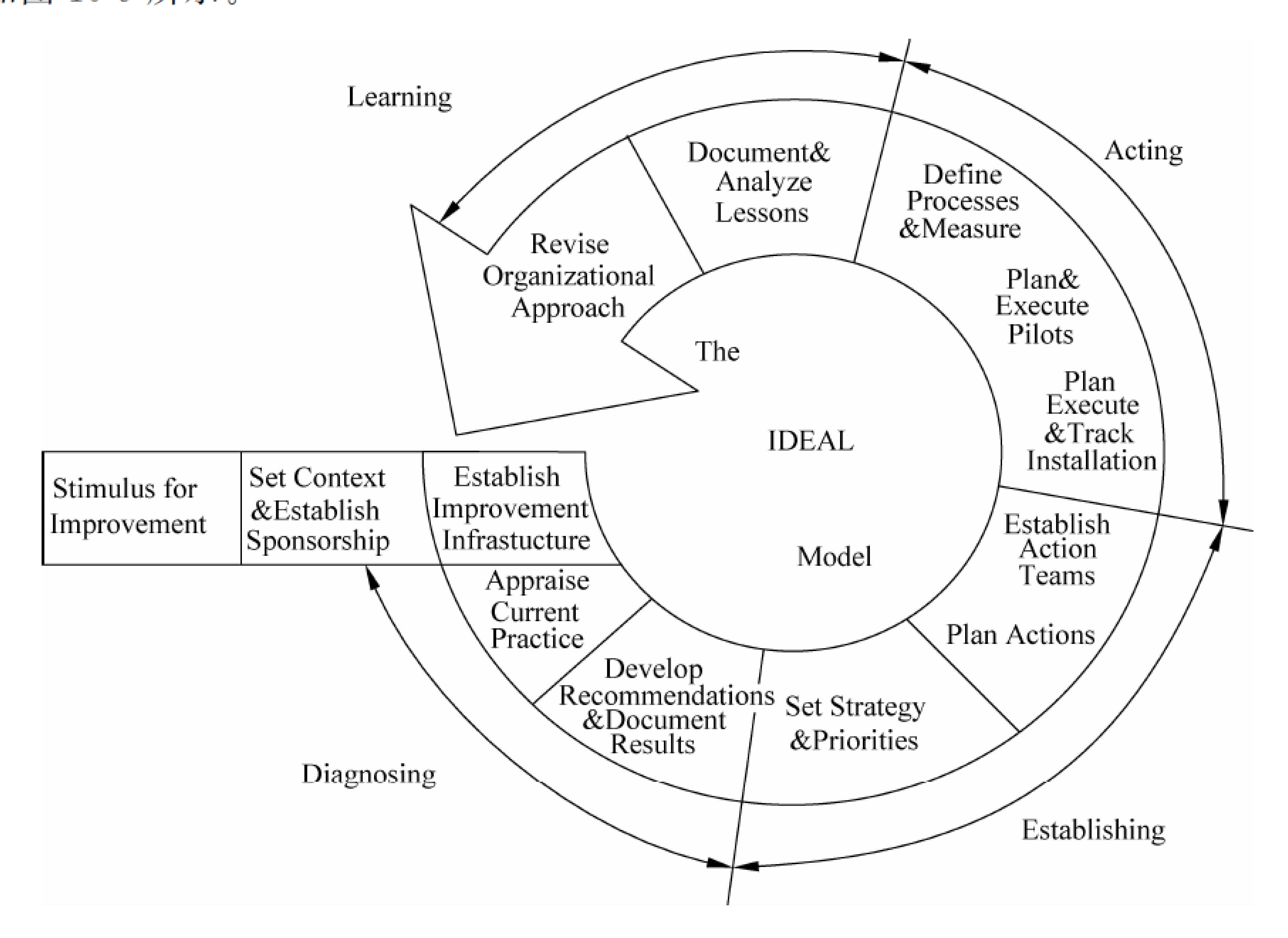


图 10-9 IDEAL 模型

1）启动阶段

该阶段要学习过程改进、约定启动资源并创建过程基础设施。最主要的是建立管理操作组（Management Steering Group，MSG）和软件工程过程组（Software Engineering Process Group，SEPG）。常常组成一个问题发现小组（Discovery Team）研究有关议题，并将软件过程改进（Software Process Improvement，SPI）建议提交给高层管理者。启动阶段的任务如下：

- 开始
- 识别改进的商业需要和推动者
- 建立 SPI 提议
- 获得支持
- 获得 SPI 提议的批准和启动资源
- 创建软件过程改进基础设施
- 评估 SPI 的情况（Climate）
- 确定 SPI 总的目标
- 确定 SPI 程序的指导原则
- 正式启动

2）诊断阶段

该阶段用于建立当前过程的成熟度、过程说明、度量等，并启动行动计划的开发工作。诊断阶段的任务如下：

- 确定需要的基线
- 基线计划
- 指导基线
- 结果展示
- 开发最终结果和建议报告
- 与组织就最终结果和建议报告进行沟通

3）建立阶段

该阶段用于建立目标和优先权，以完成行动计划的开发工作。建立阶段的任务如下：

- 选择和得到战略规划过程的培训
- 评审组织远景（Vision）
- 评审组织的商业计划
- 确定关键商业议题
- 评审过去的改进工作
- 说明改进的动机
- 确定当前和将来的改进工作
- 确定不同基础设施实体的角色和责任
- 确定改进活动和开发议题的优先级
- 协调在基准结论和建议内已有的和计划的改进工作
- 将一般的 SPI 目标变成特定的可度量的目标
- 建立和更新 SPI 战略计划

- 建立审计、评审，并批准 SPI 战略计划，约定行动的资源
- 形成技术工作组(Technical Working Group，TWG)

4) 实施阶段

实施阶段用于研究和开发过程问题的解决方案，并将成功的过程改进扩展到整个组织。实施阶段的任务如下：

- 完成 TWG 的具体计划
- 开发解决方案
- 试验有潜力的解决方案
- 选择解决方案提供商
- 确定长期的支持需要
- 开发首次实行的战略和计划的模板
- 整理改进(模板)并交给 SEPG
- 解散 TWG
- 首次实行解决方案
- 转移到长期支持

5) 学习阶段

该阶段准备再次通过 IDEAL 模型，应用已有的经验教训完善 SPI 过程。学习阶段的任务如下：

- 收集已有的经验教训
- 分析已有的经验教训
- 修改组织方法
- 评审保证人和约定
- 建立高级的目标
- 开发新的或修改的 SPI 提议
- 继续进行 SPI

6) 管理软件过程改进

对改进过程提供监督并解决相应的问题。管理软件过程改进的任务如下：

- 设置 SPI 的阶段
- 组织 SPI 程序
- 计划 SPI 程序
- 组织 SPI 程序人员
- 监督 SPI 程序
- 指导 SPI 程序

大多数情况下 SPI 的组织基础设施由三个部分组成：软件工程过程组(SEPG)、管理指导组(MSG)和技术工作组(TWG)。

CMMI 具体实施步骤如下所示。

(1) 阶段 1：CMMI 项目启动会明确组织实施 CMMI 的商业目标，建立 CMMI 项目实施的沟通机制。

(2) 阶段 2：CMMI 基础培训和 SEPG 组建，进行 CMMI 基础概念讲解，指导组织建立

核心的过程改进小组。

(3) 阶段3：充分了解组织研发过程现状，识别组织现有软件过程与组织现阶段理应达到的CMMI成熟度级别的差距，提交诊断报告，进行过程改进的策划。

(4) 阶段4：PA培训和文件定义，结合组织过程现状进行CMMI过程域培训，通过举例、案例分析等方式，让SEPG掌握过程文件定义技巧，结合实际情况有针对性地定义组织的研发过程，并确定过程产出物(如需求报告)。

(5) 阶段5：项目试点，选择代表组织核心业务的项目或者典型项目进行试点，通过试点来完善过程文件，从而为组织全面推广过程文件打下基础。

(6) 阶段6：组织推广全员参与全面导入与执行CMMI。

(7) 阶段7：预评估验证组织推广的结果，识别组织尚存缺陷并制定再次改善方案，准备充分，以便组织能够更好地进行正式的SCAMPI (Standard CMMI Appraisal Method for Process Improvement)评估。

(8) 阶段8：SCAMPI正式评估。由SEI授权的主任评估师领导，采用SCAMPI评估方法，对组织的能力成熟度进行正式评估，颁发证书，通过SEI网站向全球发布组织信息。

在CMMI的实施中将强调以下几点：

- 必须制定计划，一切按计划执行
- 应制定相应的规程，使所有事件可被重复而与人员无关
- 注重评审和验证，关键工作产品要经过正式评审
- 实现定期监控
- 要求对所有发现的问题、措施项都要进行跟踪，并且一直跟踪到结束
- 通过质量保证，保证过程得到遵守和不断地修改
- 注意采集度量和分析

实施CMMI的几个关键步骤：

- 获得管理者的有效支持
- 建立SEPG等小组，明确职责
- 组织人员培训
- 借鉴CMMI科学的系统思想，建立基本的过程体系
- 切实执行
- 建立有效的反馈机制
- 过程实施的度量和验证
- 建立过程文化

10.2　软件配置管理

10.2.1　软件配置管理概述

在软件开发项目进行中，最主要的问题之一就是持续不断的变化。这些变化可能导致项目混乱，例如在开发过程中可能会遇到如下问题：

- 找不到某个文件的历史版本

- 开发人员使用错误的程序版本
- 开发人员未经授权修改代码或文档
- 人员流动,交接工作不彻底
- 无法重新编译软件的某个历史版本
- 因协同开发,或者异地开发,版本变更混乱导致整个项目失败

如何管理这些变化,做到结构化、有序化、产品化地控制软件系统演变,就成为软件工程过程中必须重视的关键问题。

软件配置管理(Software Configuration Management,SCM)是一种应用于整个软件生命周期的质量保证活动,始于项目开发之初,终于产品淘汰之时。配置的概念来源于制造系统管理,目标是识别和管理组成复杂系统的各个部分——工件。随着现代软件技术的发展,对于软件项目的需求日益复杂而且变更频繁,开发模式也由昔日的手工作坊式转变为规模化、团队式的开发。当开发团队发展到一定规模时,会越来越强调开发过程规范化和成熟度,而将软件看做单一产品也面临着许多无法解决的问题。软件项目的成败在很大程度上取决于对其开发过程的控制,包括对质量、源代码、进度、资金、人员等的控制。所以,将软件分解为子系统、模块、构件等"工件"并对整个软件项目实施配置管理是富有成效的和现实的方法。

软件配置管理是一套规范、高效的软件开发管理方法,同时也是提高软件质量的重要手段。它应用技术和管理对项目进行指导和监督,标识和归档配置项的功能和物理特性,控制这些特性的变更,记录和报告变更过程以及实现状态,检查对指定需求的评价和意见。软件配置管理可以帮助开发团队对软件开发过程进行有效地变更控制,从而高效地开发高质量软件。配置管理在软件质量体系的诸多支持活动中处于中心位置,它有机地将所有支持活动结合起来形成一个整体,相互促进,相互影响,有力地保证了质量体系的实施。

软件配置管理并不是一个新概念,早在 20 世纪 70 年代,加州大学圣巴巴拉分校(University of California at Santa Barbara)的 Leon Presser 教授就已经提出变更和配置控制的理论。随着软件产业的逐渐壮大,软件配置管理在国外成熟的软件企业中逐渐得到重视和普及,在国外的一些大中型软件企业中,不仅设置专职的配置管理人员,而且有些公司还设有公司级的变更控制委员会(Change Control Board,CCB)对公司的软件配置管理工作进行统一管理,配置管理对于软件开发的重要性由此可见一斑。随着 CMM 概念和理论的普及,配置管理作为 CMML2 的一个关键过程域,其重要性逐渐被人们认同。

10.2.2 软件配置管理的基本概念

1. 软件配置管理

Wayne Babich 在 *SCM Coordination for Team Productivity* 中提到,"在协调软件开发中,使混乱减到最小的技术叫做软件配置管理,它是一种标识、组织和控制修改的技术,目的是使错误达到最小并最有效地提高生产效率"。

IEEE 729 标准将软件配置管理定义为,识别和定义系统中配置项的过程,通过配置管理可以在生命周期中控制配置项的变更,记录并报告配置项及变更需求的状态,检验配置项的完整性和正确性。这个定义包括如下含义:

（1）配置管理是一个准则，可发挥技术和管理上的指导和监督作用。依据这个准则设计系统的规则，并且必须在软件配置管理计划中有详细的描述。也就是说，软件配置管理必须有组织地建立，并执行技术与管理上的监督工作。软件配置管理需要成立软件配置管理组织，其中一组人员肩负不同的责任，执行软件配置管理不同的职能；另一组人员根据规则监督软件配置管理活动。该组织的人员数量和组织结构将随着项目规模和复杂性而变化。

（2）配置管理的功能是标识配置项并将它们的功能和物理特性归档。IEEE 定义配置项是一个硬件、软件或两者的集合，用于配置管理，并在软件配置管理的过程中作为一个整体被对待。软件配置管理活动必须标出软件系统的组成，然后将它们的属性归档。其功能特性包括功能描述、性能标准、物理特性，如大小、行数、模块数、函数和库。一旦标识出配置项并将它们的属性归档，软件配置管理系统将开始控制对这些属性的变更。

（3）配置管理系统会记录变更过程，并将其内容报告给有关人员，这需要建立变更管理过程档案。变更需求的状态将自始至终被跟踪。

（4）通过一些机制检查正在开发的或已经交付的系统，软件配置管理将确保已交付的和即将交付的系统是完全符合需求定义和说明。因此，软件配置管理要具备审核和验证机制。

软件配置管理要解决的问题包括以下几个：

- 多人同时修改程序或文档
- 人员流动
- 软件维护中的历史重现
- 控制软件的复杂性
- 影响项目进度的特殊因素
- 已修复的错误仍然存在
- 协同开发中的重复工作

实施软件配置管理的目的是在软件生命周期内建立和维护软件产品的完整性，保证团队的有效协作。因为软件变更在任何时刻都可能发生，因此软件配置管理活动的目标就是：

- 标识变化
- 控制变化
- 确保适当地实现了变化
- 向需要知道这类信息的人报告变化

软件配置管理由适于所有软件开发项目的最佳工程实践组成，通过以下手段来提高软件件的可靠性和质量：

- 在整个软件生命周期中提供标识和控制文档、源代码、接口定义、数据库等工件机制。
- 提供满足需求、符合标准、适应项目管理及其他组织策略的软件开发和维护的方法学。
- 为管理和产品发布提供支持信息，如基线状态、变更控制、测试、发布、审计等。

2. 软件配置项（Software Configuration Item，SCI）

所有在软件过程中产生的信息，被统称为软件配置项（见表 10-2），主要包括：

- 计算机程序(源代码和可执行程序)
- 描述计算机程序的文档(供技术人员和用户使用)
- 数据(包含在程序内部或外部的)

表 10-2 配置项的内容

配 置 项	包 含 内 容	
项目管理过程文档	项目任务书	个人日报和周报
	项目计划	项目会议纪要
	项目周报	培训记录和培训文档
QA 过程文档	QA 不符合报告	
	QA 周报	
	评审记录	
工作产品	需求文档	测试文档
	设计文档	软件说明书和手册
	代码	
第三方产品	例如：Oracle、Java 等	

一个软件系统划分为几个配置项要由项目经理所确定的开发策略决定。每个软件或某集合符合如下条件之一,可被视为一个软件配置项：

- 是独立设计、实现和测试的
- 对总体性能是关键的,或存在高风险的,或关系到系统安全性
- 极为复杂,涉及高新技术,或有严格的性能要求
- 与其他现有软件项目或由其他机构提供的软件之间有直接接口
- 预计在成为可运行软件之后会有比常规更多的修改
- 包括了某个特定范围的所有功能,如应用软件、操作系统等
- 安装在与系统其他部分不同的计算机平台上
- 被设计成可重复使用的

每个配置项的主要属性有：名称、标识符、文件状态、版本、作者、日期等。所有配置项都被保存在配置库里,确保不会混淆、丢失。配置项及其历史记录反映了软件的演化过程。

软件配置管理包含的软件配置项包括以下一些：

- 系统规格说明书
- 软件项目规划
- 软件需求规格说明书
- 设计规格说明书(数据设计、体系结构设计、模块设计、接口设计和对象描述(使用面向对象技术时))
- 源代码清单
- 测试计划和过程,测试用例和测试结果记录
- 操作和安装手册
- 可执行程序(可执行程序模块和连接模块)
- 数据库描述(模式、文件结构和初始内容)
- 用户手册

- 维护文档(软件问题报告、维护请求和工程变更次序)
- 软件工程标准
- 项目开发小结

对配置库的组织可采用面向对象的方法：将每个软件配置项看做一个配置对象，有自己的名字和属性，各配置项之间的关系用对象间关系表示，如图 10-10 所示。

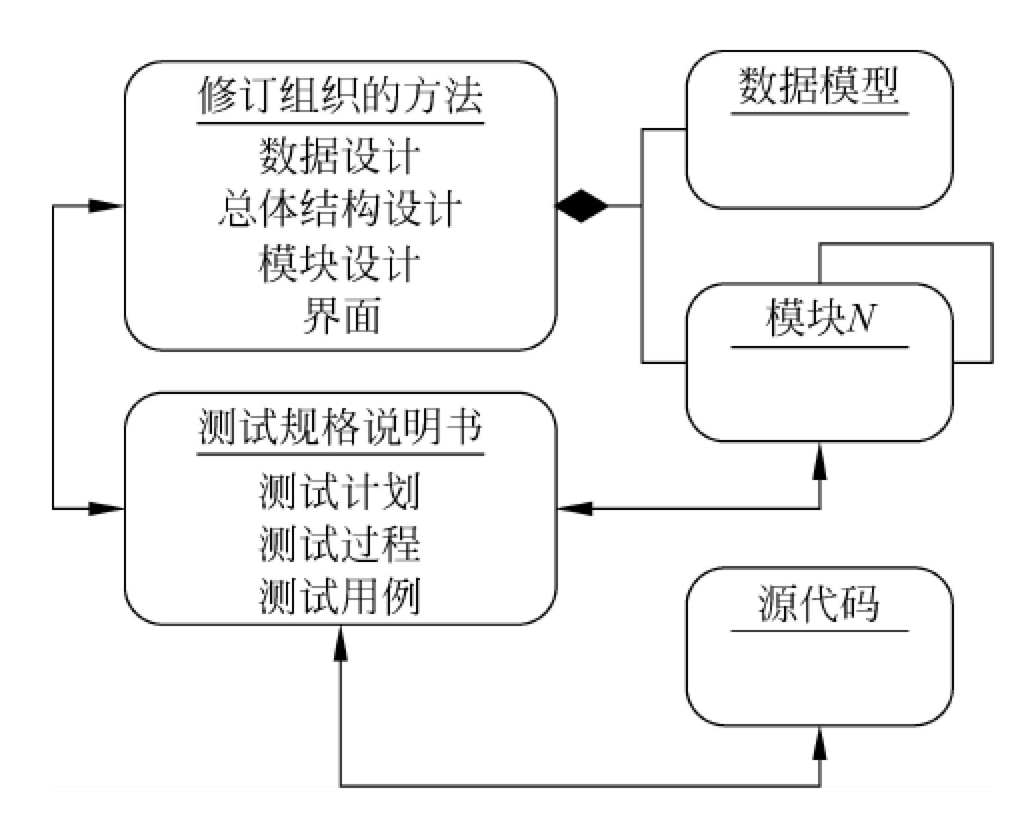


图 10-10 SCI 库的组织

软件配置项标识是管理配置的前提。软件中每个组件的标识必须唯一，以便于用该标识符来跟踪和报告软件配置项的状态。项目组人员按照标识规则对配置项进行标识，最后提交给配置管理员纳入配置库统一管理。通常，对每一个软件配置项要赋予一个标识名称或符号，软件配置项的各部分又在该标识符下附上描述符。例如：组成航天飞机飞行软件的软件配置项可标识为 FS，而该飞行软件的组成部分，如飞行执行程序可标识 FS-EX，表示它是 FS 软件配置项的第二层组件；该执行程序的各元件可编号为 FS-EX001、FS-EX002 等。因此，可以根据“型号代号-分系统/设备配置代号-所处研制阶段代号-软件产品分类编号-配置项编号”原则来对各软件配置项及其组件、子程序和相关描述文档进行命名、编号。

在配置项命名时应注意以下两点。

(1) 唯一性：在一个项目内不能出现重名的配置项。

(2) 可追溯性：名称应能体现相邻配置项之间的关系。

3. 基线(Baseline)

配置项的识别是软件配置管理活动的基础，也是制定配置管理计划的重要内容。软件配置项分类软件的开发工作是一个不断变化的过程，为了在不严重阻碍合理变化的情况下来控制变化，软件配置管理引入了“基线”概念。IEEE 对基线的定义是：基线是已经通过正式技术评审的某种产品，可以作为进一步开发的基础，并且只能通过正式的变更控制而改变。

基线指一个配置项在其生命周期的某一特定时间，被正式标明、固定并经正式批准的版本。也可以说，基线是软件生命周期中各开发阶段末尾的特定点，又称里程碑。只有由正式的技术评审而得到的软件配置项协议和软件配置的正式文本才能被称为基线。它的作用是使各阶段工作的划分更加明确化，使本来连续的工作在这些点上断开，以便于检验和肯定阶

段成果。例如，明确规定不允许跨越基线修改另一阶段的文档。

根据基线的定义，在软件的开发流程中可把所有需加以控制的配置项分为两类：基线配置项和非基线配置项。前者指经过正式评审和认可的一组软件配置项，可作为下一步软件开发工作的基础，并且只有通过正式的变更控制流程才能被更改；后者是指没有正式评审或认可的一组软件配置项。例如：基线配置项可能包括所有的设计文档、源程序等；非基线配置项可能包括项目的各类计划、报告等。

任意一个软件配置项，一旦形成文档并审核通过，便形成了一个基线，可以作为一个检查点。在软件开发过程中，当采用的基线发生错误时，可以知道其所处的位置，返回到最近的和最恰当的基线上。

基线可分为以下四类：

(1) 功能基线(Functional Baseline)，如系统分析和软件定义阶段的系统规格说明。

(2) 指派基线(Allocated Baseline)，如软件需求分析阶段的需求规格说明。

(3) 产品基线(Production Baseline)，如组装和测试阶段有关产品的规格说明。

(4) 其他基线。

常用的软件基线如图 10-11 所示。

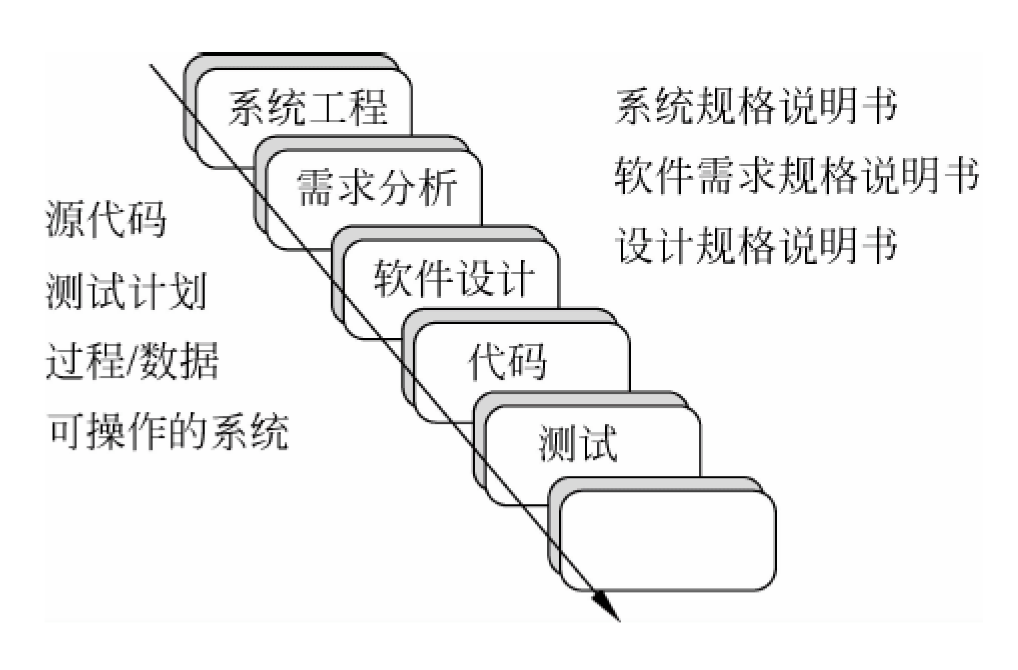


图 10-11 常用的软件基线

软件过程中的配置基线如图 10-12 所示。

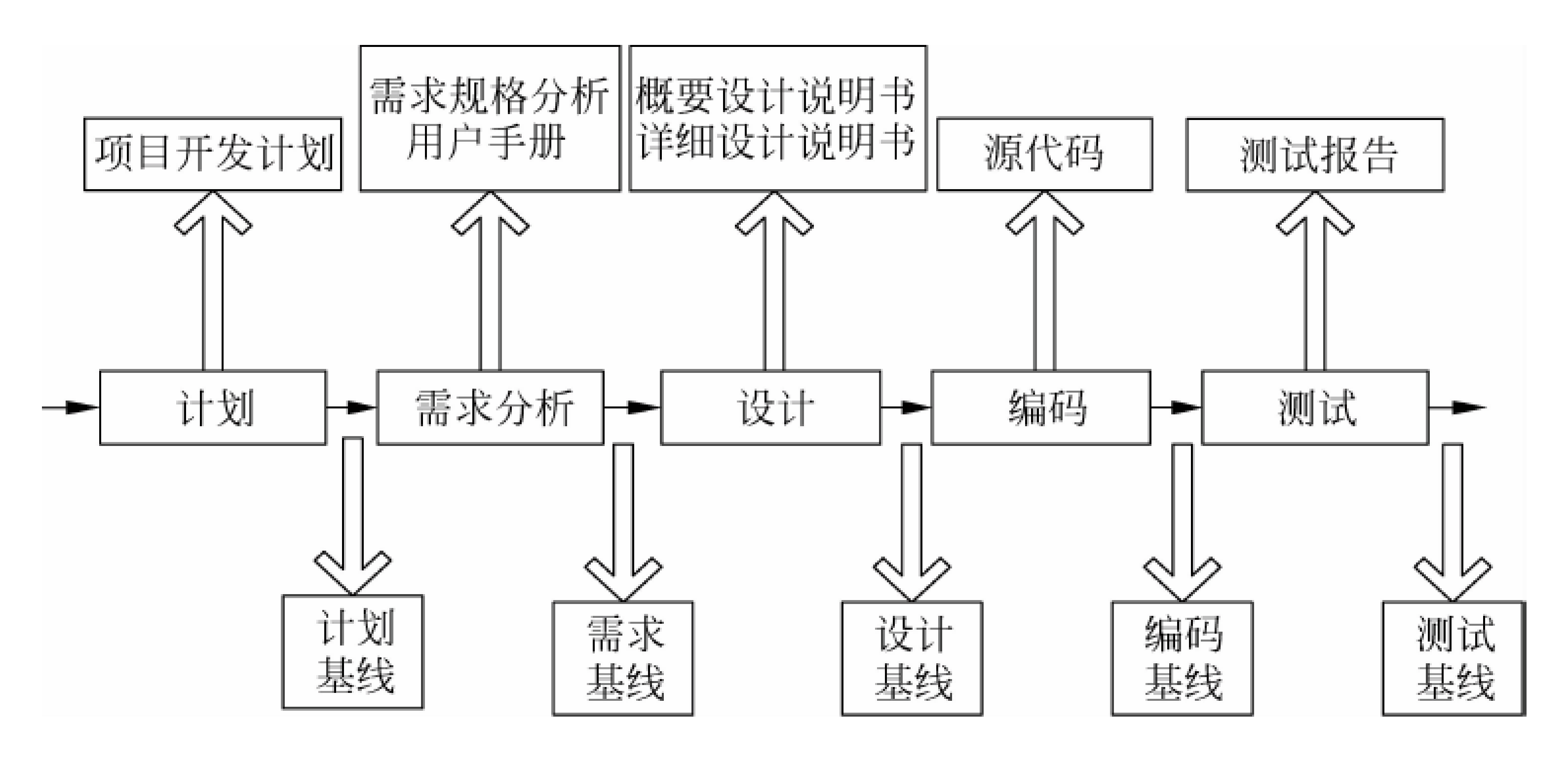


图 10-12 软件配置过程中的基线

基线的优点有以下三个。

(1) 重现性：当更新不稳定或不可信时，基线提供一种取消变更的方法。

(2) 可追溯性：建立项目工件之间的前后继承关系。

(3) 版本隔离：新项目与随后对原始项目所进行的变更进行隔离。

软件配置项一旦成为基线，就被放入项目数据库中。若要修改基线，首先要将其复制到私有工作区并在项目数据库中锁定，不允许他人使用。在私有工作区完成修改控制过程并通过复审后，再将修改后的软件配置项推出并送至项目数据库，同时解锁。

4. 配置管理工具

软件配置管理最早是使用人工的方法，以类似档案管理的方式管理软件配置项。这种管理方式工作效率极低，特别是当软件较大时，对大量的文档进行更改控制、配置审计工作，很容易出错。随着管理水平的提高，出现了用计算机进行管理的软件配置管理工具。

软件配置管理工具支持用户对源代码清单的更新管理，以及对重新编译与链接代码的自动组织；支持用户在不同文档的相关内容之间进行相互检索，并确定同一文档某一内容在本文档中的设计范围；同时还支持软件配置管理小组对软件配置更改进行科学管理。

第一代软件配置管理工具可以说仅仅是处理文件版本控制的工具。它们是基于单一文件的工具，将各独立文件的改变存储在特殊的文档文件中，一般支持恢复提交模式，并提供分支。最早出现的这类工具是 SCCS 和 RCS。这两种工具是软件配置管理的鼻祖，许多软件配置管理工具都将它们作为文件归档的工具。这一代中的其他商用产品有 Sun 公司开发的 Team Ware。第二代配置管理工具则是基于项目数据库的，支持并行开发团队协作以及过程管理。这一代工具的最显著特征是软件开发项目的源代码与它们的文档分离，且存储在一个数据库中，该数据库成为项目数据库或软件库。这种结构将重点从文件级移到了项目级，并对整个项目信息有一个统一的观点。这一代配置管理工具有基于变动请求的 IBM 的 CMVC，面向操作的 Platium 公司的 CCC 以及 SQL 公司的 PCMS 等。第三代配置管理工具，全面结合各个软件开发环节的软件配置管理整体解决方案，在保持了第二代配置管理工具优点的基础上加入了"文件透明性"这一特性。最具有代表性的产品是 Rational ClearCase，它是通过一个独占的文件系统 MVPS 来实现文件透明性的。近二三十年来，软件配置管理的任务和作用始终没有改变过。唯一改变的是那些以软件配置管理为核心的配置管理工具及其操作系统。这些工具已经从简单的版本控制和半自动构造系统进化到现在复杂的软件配置管理，通过这些工具，实现了人工无法达到的功能，真正做到了自动软件配置管理。

软件配置管理工具的功能有以下几个。

1) 权限控制(Access Control)

权限控制对软件配置管理工具来说至关重要。一方面，既然是团队开发，就可能需要限制某些成员的权限，特别是大项目往往牵扯到子项目外包，到最后联调阶段会涉及很多不同的单位，更需要权限管理。另一方面，权限控制也减小了误操作的可能性，间接提高了软件配置管理工具的可用性。

现有的软件配置管理工具，虽然在权限控制方面差异很大，但也存在共性：其核心概念是行为主体、行为客体、行为。

- 行为主体：即用户(User)。用户组(User Group)并不是行为主体，但它的引入大大方便了权限管理。

• 行为客体：即项目和项目成员(Member)。不管从软件配置管理工具的开发者还是使用者的角度，项目和项目成员都是不同的行为客体。

• 行为：即由主体施加在客体之上的特定操作，签入和签出是最典型的例子。

确定了三个核心概念之后，权限可被定义为一个四元向量：(主体，客体，行为，布尔值)，即主体在客体上施加某种行为是否被获准。

2) 版本控制(Version Control)

软件配置管理工具记录项目和文件的修改轨迹，跟踪修改信息，使软件开发工作以基线渐进的方式完成，从而避免了软件开发不受控制的局面，使开发状态变得有序；可以对同一文件的不同版本进行差异比较，恢复个别文件或整个项目的早期版本，使用户方便地得到升级和维护所必需的程序和文档；其内部对版本的标识，采用了版本号(Version Number)方式，但对用户提供了多种途径来标识版本，被广泛应用的有：版本号、标签(Label)和时间戳(Time Stamp)。多样灵活的标识手段，为用户提供了方便。

3) 增强的版本控制(Enhanced Version Control)

快照(Snapshot)和分支(Branch)以基本的版本控制功能为基础，使版本控制的功能又更进一步增强。快照是比版本高一级的概念，它是项目中多个文件各自的当前版本的集合。快照使恢复项目的早期版本变得方便，它还支持批量签入、批量签出和批量加标签等操作。总之，快照是版本控制的一种增强，使版本控制更加方便高效。

分支是版本控制的另一种增强，允许用户创建独立的开发路径。其典型用途有：分支和合并一起，是并行开发(Concurrent Development)的有力支持；分支支持多版本开发，这对发布后的维护尤其有用。比如客户报告有打印 bug，小组可能从某个还未引入打印 bug 的项目版本引出一个分支，最终发布一个 bug 修订版。

版本控制和增强的版本控制是软件配置管理工具其他功能的基础。

4) 变更管理(Change Management)

软件配置管理工具提供有效的问题跟踪(Defect Tracking)和系统变更请求(System Change Requests，SCRs)管理。通过对软件生命周期各阶段所有的问题和变更请求进行跟踪记录，来支持团队成员报告(Report)、抓取(Capture)和跟踪(Track)与软件变更相关的问题，以此了解谁改变了什么，为什么改变。

变更管理有效地支持了不同开发人员之间，以及客户和开发人员之间的交流，避免了无序和各自为政的状态。

5) 独立的工作空间(Independent Workspaces)

开发团队成员需要在开发项目上协同、并发地工作，这样可以大大提高软件开发效率。沙箱(Sandbox)为并行开发提供了独立的工作空间，在有的软件配置管理工具中也称为工作目录(Working Folder)。

使用沙箱，开发人员能够将所有必要的项目文件复制到一个私有的树形目录，修改在这些副本上进行。一旦对修改感到满意，就可以将修改合并到开发主线(Main Line)上去；如果该文件只有该成员一人修改，只需要将修改过的文件签入到主项目中即可。

并发和共享是同一事物的不同方面，并发的私有工作空间共享同一套主项目(Mater Project)文件，因此有必要让所有团队成员拥有得知项目当前状态的能力。软件配置管理工具提供刷新(Refresh)操作：某位团队成员可以使其他团队成员在主项目文件上所做的

变更，在自己沙箱的图形用户界面上反映出来。

6）报告(Report)

为保证项目按时完成，项目经理必须监控开发进程并对发生的问题迅速做出反应。报告功能使项目经理能够随时了解项目进展情况；通过图形化的报告，开发的瓶颈可以一目了然地被发现；标准的报告提供常用的项目信息，定制报告功能保证了拥有适合自己需求的信息。

7）过程自动化(Process Automation)

软件配置管理工具使用事件触发机制(Event Trigger)，即让一个事件触发另一个事件产生行为，来实现过程自动化。比如，让“增加项目成员”操作自动触发“产生功能描述表”操作，开发人员填制该文件的功能描述表，规范开发过程。

过程自动化不仅可以缩短复杂任务的时间，提高生产率，而且还规范了团队开发的过程，减少了混乱。

8）管理项目的整个生命周期

从开发、测试、发布到发布后的维护，软件配置管理工具的使命贯穿整个软件生命周期。软件配置管理工具应预先提供典型的开发模式的模板，以减少用户的劳动；另一方面，也应支持用户自定义生命周期模式，以适应特殊开发需要。

9）与主流开发环境的集成

将版本控制功能与主流集成开发环境(IDE)集成，极大地方便了软件开发过程。从集成开发环境的角度看，版本控制是其一项新功能；从软件配置管理工具的角度看，集成开发环境充当了沙箱的角色。

实施软件配置管理所面临的第一步就是要选择合适的工具，在此列出一个成熟的软件配置管理工具应该具备的特征。

- 配置项(对象)管理：版本控制、配置管理、并行开发支持、基线支持。
- 构建与发布管理：能利用流行的构建工具、支持多平台构建、支持并行构建、能自动处理构建依赖关系、能收集和维护重新产生之前构建所需要的信息。
- 工作空间管理：能自动跟踪工作空间中所有类型的变更、能应用不同配置填充工作空间、工作空间既允许隔离又允许更新。
- 流程管理：不同类型的对象都应具备流程定制能力、流程的范围可定制、支持测试与发布流程。
- 分布式开发的支持：负载均衡。
- 与其他工具的集成能力：包括变更请求工具、开发工具、其他 CASE 工具、命令行及 SDK。
- 易用性和易管理性。

常用的软件配置管理工具有如下几种。

1）面向工程的配置管理系统(CCC/HAVEST)

CCC/HAVEST 是 CA 公司的产品，是一个基于团队开发的，提供以过程驱动为基础的，包含版本管理、过程控制等功能的配置管理工具。CCC/HAVEST 中的 CCC 代表 Configuration and Change Control，即配置变更控制。CCC/HAVEST 可帮助用户在异构平台、远程分布，以及并行开发活动的情况下保持工作的协调和同步。不仅如此，它还可以

有效跟踪复杂的企业级开发的各种变化差异，从而使用户可以在预定的交付期限内提交高质量的软件版本。CCC/HAVEST 能确保开发团队开发出支持已定义和可重复过程的软件产品，使得开发产品遵循严格的标准、过程和策略：需求-编码-测试-生成产品。

2）版本控制工具——VSS

VSS(Visual Source Safe)是 Microsoft 公司推出的配置管理工具，是 VisualStudio 的套件之一。SourceSafe 是国内最流行的配置管理工具，其优点可以用八个字来概括——“简单易用，一学就会”。它基于客户-服务器结构，在服务器端建立 VSS 的数据库，共享该数据库，客户端指定连接到该数据库，并且支持用户级管理，对中文的支持也比较好。VSS 使用反响增量技术，确保一个文档的所有版本都是可用的，并使用不同的机制存储文本文件和二进制文件。在实际使用中，VSS 提供了在网络应用系统开发中的文件共享和文件锁定特性，可确保团队开发中代码的完整性和一致性。它可以使开发人员对源代码和由 Visual J++、Visual Basic、Visual C++ 和 Visual Foxpro 开发的部件进行管理，对软件版本开发进度进行管理和控制，并可以防止由于网络文件锁定导致的版本冲突。此外，VSS 还可以与 Visual InterDev 紧密集成，管理动态 Web 应用系统中的各种部件，这样可以大大提高团队开发中的进度管理的有效性。

3）CVS

CVS 是并行版本系统(Concurrent Version System)的缩写，它是开放源代码的配置管理工具。与 VSS 相比，CVS 的主要优点有以下三个：

（1）VSS 有的功能 CVS 全都有，而且 CVS 支持并发的版本管理，VSS 没有并发功能。CVS 服务器的功能和性能都比 VSS 高出一筹。

（2）CVS 服务器是用 Java 编写的，可以在任何操作系统和网络环境下运行。CVS 深受 UNIX 和 Linux 的用户喜爱。Borland 公司的 JBuilder 提供了 CVS 的插件，Java 程序员可以在 JBuilder 集成环境中使用 CVS 进行版本控制。

（3）CVS 服务器有自己专用的数据库，文件存储并不采用 VSS 的“共享目录”方式，所以不受限于局域网，信息安全性很好。

CVS 的主要缺点：客户端软件五花八门、良莠不齐。UNIX 和 Linux 的软件高手可以直接使用 CVS 命令行程序，而 Windows 用户通常使用 WinCVS。安装和使用 WinCVS 显然比 VSS 麻烦不少，这是比较令人遗憾的。

4）ClearCase

Rational 公司的 ClearCase 是软件行业公认的功能最强大、价格最昂贵的配置管理软件，主要应用于复杂产品的并行开发、发布和维护，其功能划分为四个范畴：版本控制、工作空间管理、构造管理和过程控制。ClearCase 通过 TCP/IP 来连接客户端和服务器，其拥有的浮动 License 可以跨越 UNIX 和 Windows NT 平台被共享。

ClearCase 的功能比 CVS 和 VSS 强大得多，但是其用户量却远不如 CVS 和 VSS 的多。主要原因是：

- ClearCase 价格昂贵，如果没有批量折扣的话，每个 License 大约五千美元。
- 用户只有经过几天的培训后(费用同样很昂贵)，才能正常使用 ClearCase。如果不参加培训的话，用户基本上不可能无师自通。

10.2.3 软件配置管理的内容

为了及时地确定软件的配置，系统地控制软件配置的变更，保证整个软件生命周期内软件配置的完整性和可追溯性，软件配置管理可以提炼为三个方面的内容：版本控制(Version Control)、变更控制(Change Control)和过程支持(Process Support)。

1. 版本控制

版本控制是全面实行软件配置管理的基础，可以保证软件技术状态的一致性。软件开发人员在日常工作中都在或多或少地进行版本管理的工作。比如有时为了防止文件丢失，而复制一个后缀为.bak 或日期的备份文件，当文件丢失或被修改后可以通过该备份文件恢复。版本控制是对系统不同版本进行标识和跟踪的过程，其目的是便于对版本加以区分、检索和跟踪，以表明各个版本之间的关系。一个版本是软件系统的一个实例，在功能上和性能上与其他版本有所不同，或是修正、补充了前一版本的某些不足。实际上，对版本的控制就是对版本的各种操作控制，包括签入签出控制、版本的分支和合并、版本的历史记录和版本的发行。

2. 变更控制

进行变更控制是至关重要的，但是要实行变更控制也是一件比较困难的事情。我们要关注变更的发生，是因为对代码的一点小干扰就有可能导致一个巨大的错误，但是它也许能够修补一个巨大的漏洞或者增加一些很有用的功能。

3. 过程支持

现在人们已意识到了软件工程过程概念的重要性，而且也逐渐了解了这些概念和软件工程支持技术的结合，尤其是软件过程概念与配置管理有着密切的联系，因为配置管理理所当然地可以作为一个管理变更的规则(或过程)。但是，传统意义上的软件配置管理主要着重于对软件的版本管理，缺乏软件过程支持的概念。在大多数有关软件配置管理的定义中，也并没有明确提出配置管理需要对过程进行支持的概念。因此，不管软件的版本管理得多好，如果组织之间没有连接关系，组织所拥有的是相互独立的信息资源，就会导致信息的孤立。在配置管理提供了过程支持后，它与 CASE 环境进行了集成，组织之间通过过程驱动建立一种单向或双向的连接。而开发人员或测试人员却不必去熟悉整个过程，也不必知道整个团队的开发模式，他们只需要集中精力关心自己所需要进行的工作。在这种情况下，可以延续其一贯的工作程序和处理办法。

10.2.4 软件配置管理的功能

软件配置管理可分为四大功能领域。

1. 配置标识

配置标识又称为配置需求，包括标识软件系统的结构，标识独立部件，并使它们成为可

访问的。配置标识的目的是在整个生命周期中标识系统各部件,并提供对软件过程及其软件产品的跟踪能力。

2. 配置变更控制

配置变更控制包括在软件生命周期中控制软件产品的发布和变更。发布通常体现为版本管理,变更体现为变更控制,目的都是建立确保软件产品质量的机制。它可回答以下问题:什么是受控的?受控产品怎样变更?谁控制变更?何时接受、恢复、验证变更?

3. 配置状态统计

配置状态统计包括记录和报告变更过程,目标是不间断记录所有基线项的状态和历史,并进行维护。它可回答以下问题:系统已经做了什么变更?此问题将会对多少个文件产生影响?配置变更控制针对软件产品,而配置状态统计针对软件过程。因此,两者的统一就是对软件开发(产品、过程)的变更控制。

4. 配置审核

配置审核将验证软件产品的构造是否符合需求、标准或合同的要求,目的是根据软件配置管理的过程和程序,验证所有的软件产品已经产生并有正确标识和描述,所有的变更需求都已解决。它可回答以下问题:系统和需求是否吻合?是否所有变更都是在版本控制下的?

10.2.5 软件配置管理的流程

一个软件研发项目一般可以划分为三个阶段:计划阶段、开发阶段和维护阶段。然而,从软件配置管理的角度来看,后两个阶段所涉及的活动是一致的,所以可将它们合二为一,称为项目开发维护阶段。根据这样的划分,通常软件配置管理的流程如图 10-13 所示。

1. 人员确定与分工

1) PM——项目经理

项目经理(Project Manager)是负责项目管理的专业人员,负责一个项目的计划、执行及结束关闭,主要对项目目标的完成负责。项目目标包括项目的范围、成本、进度、质量、沟通等多维目标,项目经理通过专业努力,组织团队按项目要求,在一定的时间内完成项目规定的任务。

2) CCB——配置控制委员会

配置控制委员会(Configuration Control Board)实施整体变更控制。CCB 可以由一个小组担任,也可以由多个不同的小组担任,负责做出决定究竟将哪些已建议需求变更或新产品特性付诸应用。典型的变更控制委员会会同样决定在哪一些版本中纠正哪些错误。

CCB 由项目所涉及的多方成员共同组成,能代表变更涉及的团体,其成员可能包括如下几个方面的代表:

- 产品或计划管理部门
- 项目管理部门

- 开发部门
- 测试或质量保证部门
- 市场部或客户代表
- 制作用户文档的部门
- 技术支持部门
- 帮助桌面或用户支持热线部门
- 配置管理部门

CCB 是决策机构，不是作业机构。通常 CCB 的工作是通过评审手段来决定项目是否能变更，但不提出变更方案。CCB 的作用有：批准配置项的标识，以及信息系统的基线建立；制定访问控制策略；建立更改基线的设置，审核变更申请；根据配置管理员的报告决定相应的对策。

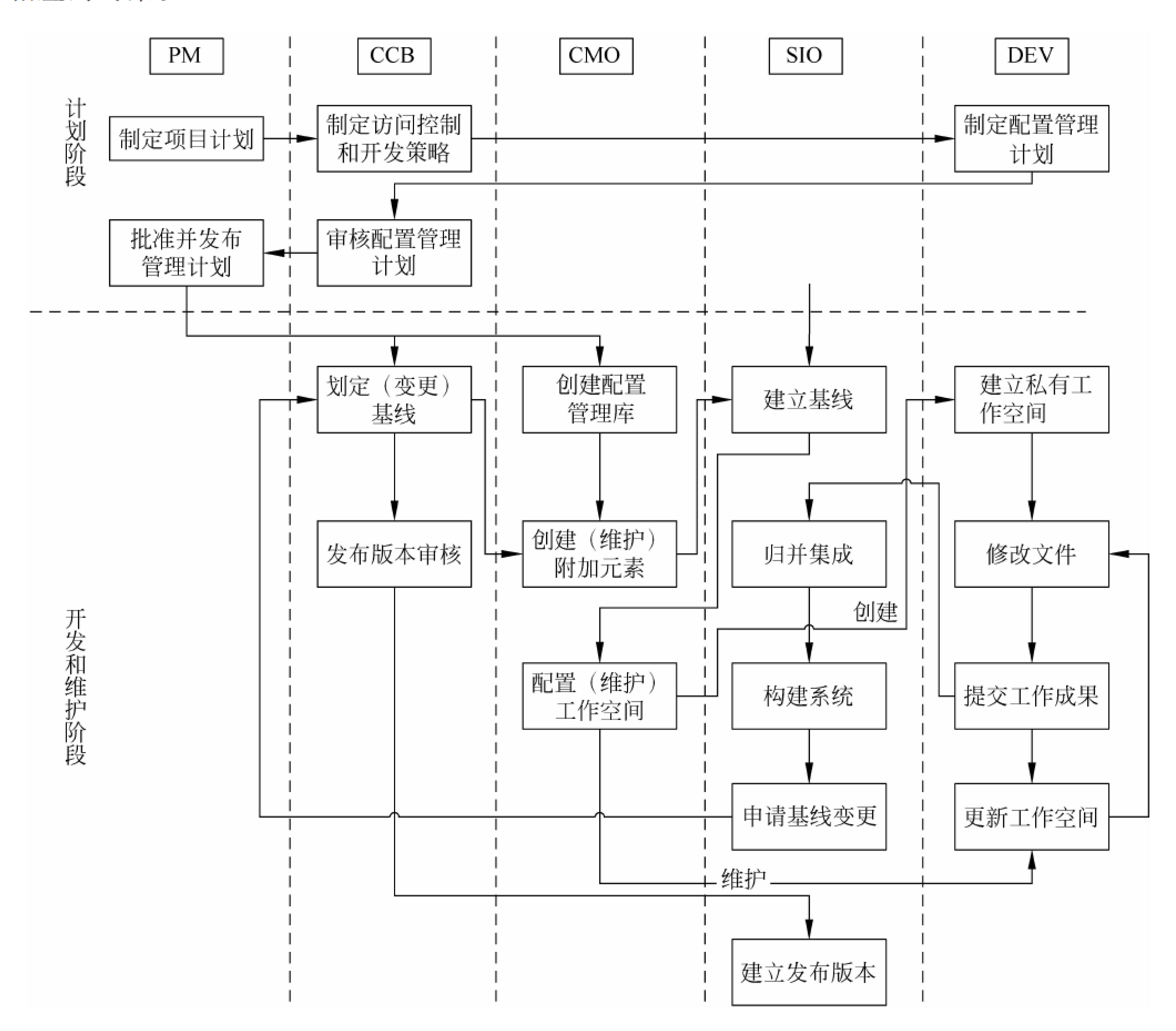


图 10-13 软件配置管理流程

3）CMO——配置管理员

配置管理员（Configuration Management Officer）根据配置管理计划执行各项管理任务，定期向 CCB 提交报告，并列席 CCB 的例会。其具体职责为以下几项：

- 文件配置管理工具的日常管理与维护
- 各配置项的管理与维护
- 执行版本控制和变更控制方案
- 完成配置审计并提交报告
- 对开发人员进行相关的培训
- 识别软件开发过程中存在的问题并拟定解决方案

4）SIO——系统集成员

系统集成员(System Integration Officer)负责生产和管理项目的内部和外部发布版本，其具体职责为：

- 集成修改
- 构建系统
- 完成对版本的日常维护
- 建立外部发布版本

5）DEV——开发人员

开发人员(Developer)的职责就是根据组织内确定的软件配置管理计划和相关规定，按照软件配置管理工具的使用模型来完成开发任务。

2. 项目计划阶段的管理

初期，项目经理需要制订整个项目的计划，它是项目研发工作的基础。在制订了项目计划之后，就可以开始制订软件配置管理计划了。如果不在项目开始之初制订软件配置管理计划，那么软件配置管理的许多关键活动就无法及时有效地开展，而它的直接后果就是造成项目开发状况的混乱，软件配置管理活动注定会成为一种“救火”的行为。所以，及时制订一份软件配置管理计划在一定程度上是项目成功的重要保证。

制订软件配置管理计划的主要过程如下所示：

(1) CCB根据项目开发计划制订访问控制和开发策略。

(2) DEV根据CCB的规划，制定详细的配置管理计划，并提交CCB审核。

(3) CCB通过配置管理计划后，交项目经理批准、发布实施。

3. 项目开发维护阶段的管理

这一部分是项目研发的主要阶段，在此软件配置管理活动主要分为如下几个层面：

(1) 主要由CMO完成管理和维护工作。

(2) CMO根据软件配置管理计划创建配置管理库、附加元素和配置工作空间，为执行软件配置管理做好准备。

(3) SIO按照项目的进度归并集成组内开发人员的工作成果，并构建系统，推进版本的演进。

(4) DEV按照统一的软件配置管理策略，根据获得的授权资源进行项目的研发工作。

(5) CCB根据项目的进展情况，审核各种变更要求，并适时地划定新的基线，保证开发和维护工作的有序进行。

上述过程会循环往复，到指导项目结束为止。除了上述的过程之外，还涉及其他一些相

关的活动和操作：

(1) 各开发人员按照 PM 发布的开发策略或模型进行工作。

(2) SIO 负责将各分项目的工作成果归并至集成分支，用于测试或发布。

(3) SIO 可向 CCB 提出设立基线的要求，经批准后由 CMO 执行。

(4) CMO 定期向 PM 和 CCB 提交审查报告，并在 CCB 例会中报告项目在软件过程中可能存在的问题和改进方案。

(5) 在基线生效后，一切对基线和基线之前的开发成果的变更都必须经 CCB 的批准。

(6) CCB 定期举行例会，根据成员所掌握的情况、CMO 的报告和 DEV 的请求，对配置管理计划做出修改，并向 PM 负责。

10.2.6　版本控制

软件配置实际上是一个动态的概念，它一方面随着软件生命周期向前推进，软件配置项的数量在不断增多，一些文档经过转换生成另一些文档，并产生一些信息；另一方面又随时会有新的变更出现，形成新的版本。

版本控制是对系统不同版本进行标识和跟踪管理的过程，是软件配置管理的核心。它的对象是软件开发过程中涉及的所有文件系统对象，如文件、目录、链接等。其目的在于按照一定的规则保存配置项的所有版本，避免发生版本丢失或混淆等现象，跟踪控制对象的变更以防止丢失，并提供访问授权，实现并行开发。

1. 版本的访问控制和同步控制

工作区中的源文件是从库中恢复得到的一个复制文件。一般有两种工作模式：

(1) 在工作区域一旦有“读”请求，就做一次恢复操作，获得复制文件，当“读”操作结束，该复制文件将被删除。

(2) 仅当软件库中的内容发生更改时，才发生交互，而不是每次“读”操作都与软件库中的文件发生交互。

访问和同步控制的流程如图 10-14 所示，其中，签入指将软件配置项从用户的工作环境存储到软件配置库的过程。签出是将软件配置项从软件配置库中取出的过程。

2. 版本分支与合并

对版本进行分支操作的人工方法是从主版本复制一份文件，做上标记。实行版本控制之后，版本的分支是一份复制文件，复制过程和标记动作由版本系统自动完成。

版本合并通过对文件的比较来进行，有两种途径。

- 将版本 A 的内容附加到版本 B 中。
- 合并版本 A 和版本 B 的内容，形成新的版本 C。

后一种途径更容易理解，也符合软件开发的思路。

3. 版本的历史记录

文件和目录的版本演化历史可以形象地表示为图形化的版本树(见图 10-15)，每一个版本都是软件配置项(源代码、文档及数据)的一个收集。

• 版本树由版本依次连接形成，每个结点代表一个版本，根结点是初始版本，叶结点代表最新的版本。
• 典型的软件系统包含多个文件和目录，每个文件和目录都有自己的版本树。
• 版本的历史记录有助于对软件配置项进行审计，有助于追踪问题的来源。
• 版本的历史记录应该包含版本号、修改时间、修改者、修改描述等内容。

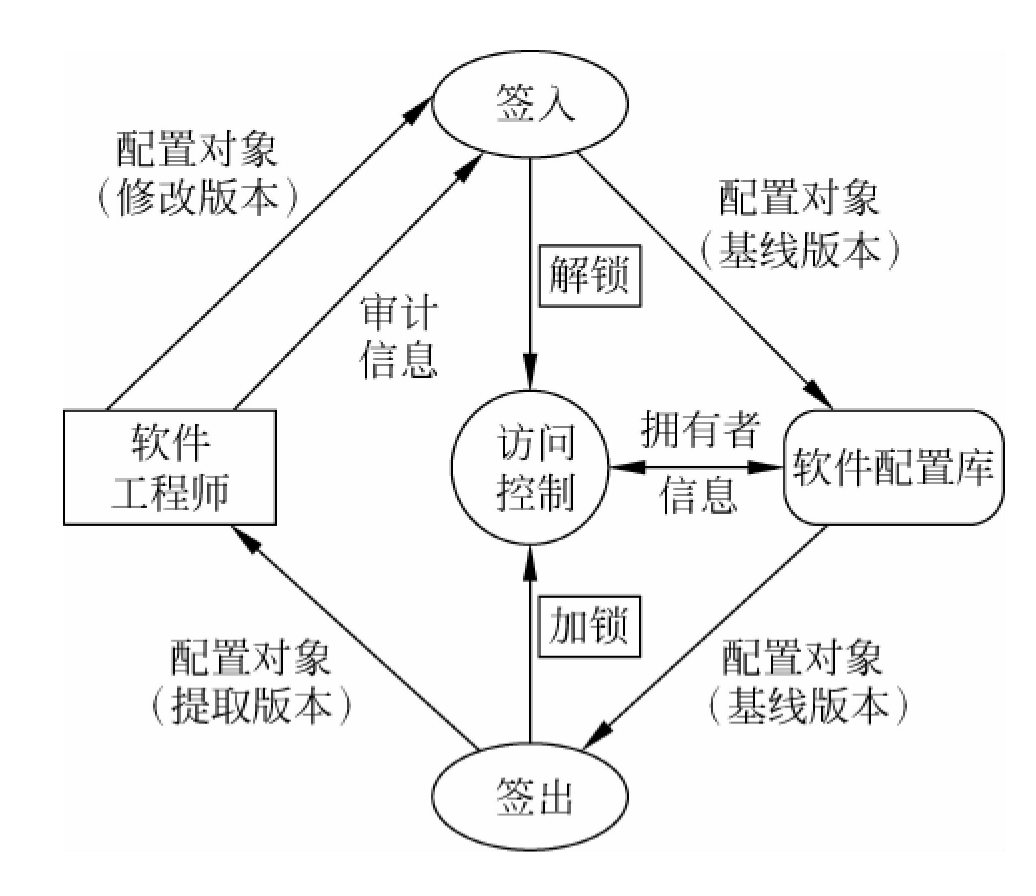


图 10-14 访问和同步控制流程图

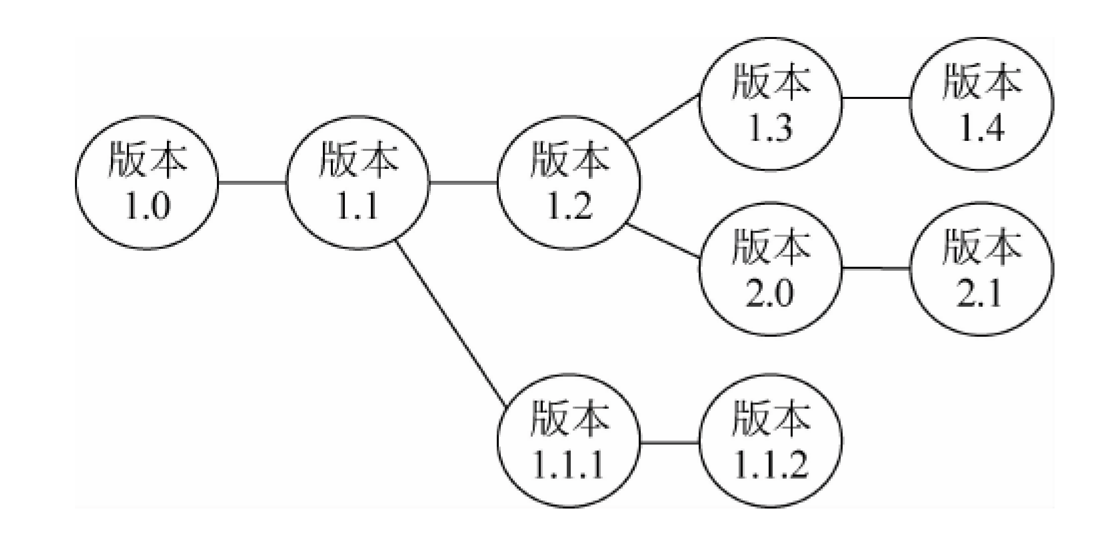


图 10-15 版本树

4. 版本号

(1) 处于“草稿”状态的配置项的版本号格式为：0.*YZ*。
• *YZ* 数字范围为 01～99。
• 随着草稿的不断完善，*YZ* 的取值应递增，其初值和增幅由用户自己把握。
(2) 处于“正式发布”状态的配置项的版本号格式为：*X.Y*。
• *X* 为主版本号，取值范围为 1～9；*Y* 为次版本号，取值范围为 1～9。
• 配置项第一次“正式发布”时，版本号为 1.0。
• 如果配置项的版本升级幅度比较小，一般只增大 *Y* 值，*X* 值保持不变。只有当配置项版本升级幅度比较大时，才允许增大 *X* 值。
(3) 处于“正在修改”状态的配置项的版本号格式为：*X.YZ*。
• 配置项正在修改时，一般只增大 *Z* 值，*X.Y* 值保持不变。
• 当配置项修改完毕，状态重新成为“正式发布”时，将 *Z* 值设置为 0，增加 *X.Y* 值，参见规则(2)。

5. 版本控制流程

- Step1：创建配置项。
- Step2：修改处于“草稿”状态的配置项。
- Step3：技术评审或领导审批。
- Step4：正式发布。
- Step5：变更。

10.2.7 变更控制

软件开发过程中会产生很多变更，如配置项、配置、基线、构建的版本、发布的版本等。变更控制是一种机制，以保证所有变更都是可控的、可跟踪的和可重现的。

1. 变更机制(见图 10-16)

CCB(配置控制委员会)负责对变更进行控制，定期开会对近期产生的变更进行分析、整理，并做出决定。

2. 变更类型

- 功能变更。功能变更是为了增加或者删除某些功能、或者为了完成某个功能的方法而需要的变更；这类变更必须经过某种正式的变更评价过程，以估计变更需要的成本和其对软件系统其他部分的影响。
- 缺陷变更。缺陷修补是为了修复漏洞需要进行的变更。在项目前期，它是必须进行的，通常不需要从管理角度对这类变更进行审查和批准。在项目后期，如果发现错误的阶段在造成错误的阶段的后面，则必须遵照标准的变更控制过程来进行。

3. 变更管理

变更管理(见图 10-17)的实施步骤如下所示。

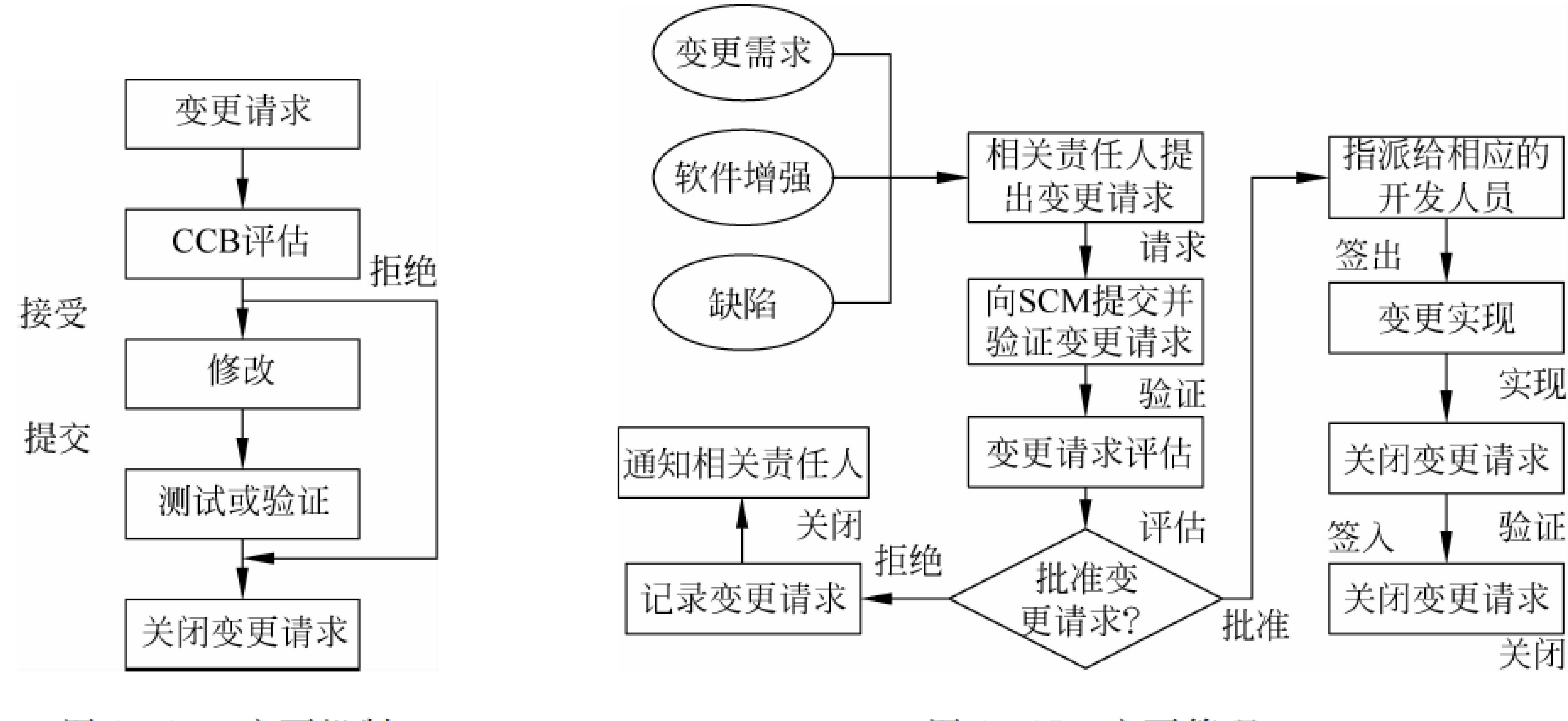


图 10-16 变更机制

图 10-17 变更管理

(1) 变更请求提交：识别变更需求，提交并记录变更申请。

(2) 变更请求接收：必须建立接收提交的变更请求并进行跟踪的机制，指定接收和处理变更请求的责任人，确认变更请求。

(3) 变更请求评估：评估变更风险、确定优先等级。

(4) 变更请求决策：制定变更计划。

(5) 变更请求实现：增强性变更需要更多的设计工作，而缺陷修复需要建立一个环境，在该环境中可以对缺陷进行重现并测试相应的解决方案。

(6) 变更请求验证：增强请求的测试通常涉及验证所做变更是否满足请求的需要，缺陷测试则简单地验证开发人员的修复是否真正消除了该缺陷。

(7) 变更请求完成：由提交请求的原有请求者中止这一循环过程。

10.3 软件质量保证

10.3.1 软件质量

ANSI/IEEE Std 729-1983 定义软件质量为“与软件产品满足需求所规定的和隐含的能力有关的特征和特性的全体”。更具体地说，软件质量是软件与明确地叙述的功能和性能需求、文档中明确描述的开发标准以及任何专业开发的软件产品都应该具有的隐含特征相一致的程度。这样的定义是从软件质量本身的特性角度描述的，即为了满足软件各项精确定义的功能、性能需求，以及软件开发符合文档化的开发标准，需要相应地给出或设计一些质量特性及其组合，作为软件开发与维护中的重要考虑因素。如果这些质量特性及其组合都能在产品中得到满足，则这个软件产品的质量是好的。这些质量特性分布在两个方面：产品质量和过程质量。产品质量是为产品本身规定的特性，而过程质量则是指软件开发过程遵守文档化开发标准程度的特性。

上述软件质量的定义强调了三个要点：

(1) 软件需求是度量软件质量的基础，与需求不一致，就是质量不高或没有质量。

(2) 指定的开发标准定义了一组指导软件开发的准则，如果不遵守这些准则，软件质量就得不到保证。

(3) 由于领域背景知识的隐含性，使得有一些隐含的需求没有被明确提出来，如软件产品的可维护性要求等，如果软件只满足那些明确描述的需求而不满足隐性需求，软件质量也是值得怀疑的。

虽然软件质量是难以定量度量的软件属性，但是仍然能够提出许多重要的软件质量指标。目前，倾向于从以下几个方面对软件质量做比较全面地评价：

(1) 软件应能按照既定的要求进行工作，在功能和性能方面都符合设计要求。系统能够可靠地工作，不仅表现为在合法输入的情况下能够正确有效地运行，还应具有处理非法输入和处理意外事件的能力，以保证应用系统不受损害。

(2) 高质量的软件应具有好的软件结构。一方面软件系统内部结构应该清晰，使软件人员易于阅读和理解，从而方便软件的修改和维护；另一方面系统的人机界面清晰、友好，便于用户使用。

(3) 软件必须文档齐全。不仅仅是可执行的程序代码,还应包括软件开发和维护过程中所产生的所有文档,这些文档资料是软件维护所不可缺少的。

因此,一个大型软件系统的质量应该从可靠性、易理解性、易维护性、效率等几个方面全面地进行评价,我们把这种评价称为宏观(抽象)标准。

优质软件的现代观点是把一个软件产品和质量因素联系起来。通过用户在使用软件产品时的三种不同倾向或观点,可把这些质量因素分成三组:产品运行、产品修改和产品转移。

1. 产品运行

(1) 正确性:系统满足规格说明和用户目标的程度,即在预定环境下能正确地完成预期功能的程度。

(2) 健壮性:在硬件发生故障、输入的数据无效或操作错误等意外环境下,系统能做出适当响应的程度。

健壮性有两层含义:一是容错能力;二是恢复能力。容错是指发生异常情况时系统不出错误的能力,对于应用于航空航天、武器、金融等领域的这类高风险系统,容错设计非常重要。恢复则是指软件发生错误后,重新运行时能否恢复到没有发生错误前的状态的能力。例如:因输入数据不正确,引起系统异常,这是容错能力不高引起的健壮性问题;操作系统死机了,重启后能够正常使用,说明具有一定恢复能力,具有一定的健壮性;数据库发生故障后,再次启动时一般能够恢复到正常的状态,说明恢复能力比较好。

(3) 可靠性:软件系统在一定的时间内无故障运行的能力。可靠性通常是由于设计中没有料到的异常和测试中没有暴露的代码缺陷引起的,是一个与时间相关的属性,指的是在一定环境下,在一定的时间段内,程序不出现故障的概率,因此是一个统计量,通常用平均无故障时间(Mean Time To Fault,MTTF)来衡量。

(4) 效率:为了完成预定的功能,系统所需计算机资源的多少。

(5) 安全性:指软件同时兼顾向合法用户提供服务,以及阻止非授权使用软件及资源的能力。

安全性既属于技术问题又属于管理问题。一般地,如果黑客为非法入侵花费的代价(考虑时间、费用、风险等多种因素)高于得到的好处,那么这样的系统就可以被认为是安全的。

(6) 可用性:产品对用户来说有效、易学、高效、好记、少错和令人满意的程度,即用户能否用软件完成他的任务,效率如何、主观感受怎样。

ISO 9241-11 将可用性定义为:产品在特定使用环境下为特定用户用于特定用途时所具有的有效性(Effectiveness)、效率(Efficiency)和用户主观满意度(Satisfaction)。其中:

- 有效性代表用户完成特定任务和达到特定目标时所具有的正确程度和完整程度。
- 效率代表用户完成任务的正确程度和完整程度与所使用资源(如时间)之间的比例。
- 满意度代表用户在使用产品过程中所感受到的主观满意程度和接受程度。

(7) 易用性:用户使用软件的难易程度。

(8) 风险:按预定的成本和进度把系统开发出来,并且用户感到满意的概率。

2. 产品修改

(1) 可理解性：理解和使用该系统的难易程度。

(2) 可维护性：为修改 bug、增加功能、提高质量而诊断并修改软件的难易程度。

(3) 灵活性：调整、修改或改进正在运行的软件系统以适应新需求、变化了的需求的难易程度。

(4) 可测试性：对软件测试以证明其满足需求规约的难易程度。

3. 产品转移

(1) 可移植性：软件不经修改或稍加修改就可以运行于不同软硬件环境的难易程度，主要体现为代码的可移植性。

(2) 可重用性：重用软件或部分软件的难易程度。

(3) 互操作性：指本软件与其他系统交换数据和相互调用服务用以协同运作的难易程度。

与其他产品相比，软件产品的质量有其明显的特殊性。

(1) 没有绝对的产品质量标准。很难制定具体的、数量化的产品质量标准，所以没有相应的国际标准、国家标准或行业标准。对软件产品而言，无法制定如“合格率”、“一次通过率”之类的质量目标。每千行的缺陷数量是通用的度量方法，但缺陷的等级、种类、性质、影响却各有不同，不能说每千行缺陷数量小的软件，一定比该数量大的软件质量更好。至于软件的可扩充性、可维护性、可靠性等，也很难量化，不好衡量。软件质量指标的量化手段需要在实践中不断总结。

(2) 没有绝对的合格界限。软件产品质量没有绝对的合格与不合格界限。软件不可能做到“零缺陷”，对软件的测试不可能穷尽所有情况，有缺陷的软件仍然可以使用。软件产品通过维护和升级来不断完善其质量。

(3) 产品之间很难比较优劣。软件产品之间很难进行横向的质量对比，很难说哪个产品比哪个产品好多少。不同软件之间的质量也无法直接比较。

(4) 满足需求的就是好的。满足了用户需求的软件质量，就是好的软件质量。客户的需求要经过双向确认，而且这种需求一开始可能是不完整、不明确的，随着开发的进行会不断进行调整。

(5) 质量的衡量标准各有不同。软件的类型不同，软件质量的衡量标准的侧重点也不同。

10.3.2 软件质量保证概述

1. 概念

与硬件系统不同，软件不会磨损；因此在软件交付之后，其可用性不会随时间的推移而改变。软件质量保证(Software Quality Assurance，SQA)就是一个系统性的工作，用于提高软件交付时的水平。它通过对软件产品和活动进行评审和审计来验证软件是合乎标准的，采用建立一套有计划性、系统性的方法，来向管理层保证拟定出的标准、步骤、实践和方

法能够正确地被所有项目所采用。

软件质量保证的目的是使软件过程对于管理人员来说是可见的，软件质量保证组在项目开始时就一起参与建立计划、标准和过程，这些将使软件项目满足机构方针的要求。软件质量保证是一种应用于整个软件开发过程的一系列系统性的保护性活动，它提供开发出满足使用要求产品的软件过程的能力证据，包括：

- 有效的软件工程技术（方法和工具）。
- 在整个软件过程中采用的正式技术复审。
- 一种多层次的测试策略。
- 对软件文档及其修改的控制。
- 保证软件遵从软件开发标准的规程、度量和报告机制。
- 为软件开发过程、产品和所使用的资源提供一个独立的视角。
- 依据标准检查产品及其文档的符合性，软件开发所使用流程的符合性。
- 通过对需求、设计和编码进行评审，减少在测试和集成阶段修改缺陷的成本。

2. 主要功能

软件质量保证就是向用户及社会提供满意的、高质量的软件产品，是确保软件产品在软件生命周期所有阶段的质量的活动，即为了确定、达到和维护所需的软件质量而进行的所有有计划性、系统性的管理活动，其主要功能为：

- 制订和展开质量方针。
- 制订质量保证方针和质量保证标准。
- 建立和管理质量保证体系。
- 明确各阶段的质量保证业务。
- 明确各阶段的质量评审。
- 确保设计质量。
- 提出、分析重要的质量问题。
- 总结实现阶段的质量保证活动。
- 整理面向用户的文档、说明书等。
- 鉴定产品质量，鉴定质量保证体系。
- 收集、分析和整理质量信息。

3. 主要任务

为了提高软件的质量，软件质量保证的任务大致可归结为如下几个。

1）正确定义用户要求

软件质量保证人员必须正确定义用户所要求的技术，重视领导全体开发人员收集和积累相关用户业务领域的各种资料和技术。

2）应用相关的技术方法

应当在开发新软件的过程中，大力使用和推行软件工程学的方法，包括标准化、设计方法论、工具化等。

3）提高软件开发的工程能力

只有拥有高水平的软件工程能力才能生产出高质量的软件产品。因此，应在软件开发环境或软件工具箱的支持下，运用先进的开发技术、工具和管理方法，提高开发软件的能力。

4）重用已有的软件

利用已有的软件成果是提高软件质量和软件生产率的重要途径。不要只考虑如何开发新软件，而应首先考虑可以重用哪些已有软件，并在开发过程中，随时考虑所生产软件的复用性。

5）发挥每个开发者的能力

软件生产是人类的智能生产活动，依赖于开发组织团队的能力。开发者必须有学习各专业业务知识、生产技术和管理技术的能动性；管理者或产品服务者要制定技术培训计划、技术水平标准以及适用于将来需要的中长期技术培训计划。

6）组织外部力量协作

一个软件自始至终由一个软件开发组织来开发也许是最理想的，但在现实中则难以做到。因此需要改善对外部协作部门的开发管理，必须明确规定进度管理、质量管理、交接检查、体制维护等各方面的要求，建立跟踪检查的体制。

7）排除无效劳动

严重的无效劳动是因需求说明有误、设计有误而造成的返工。定量记录返工工作量，收集和分析返工劳动花费的数据是非常重要的。另一种较大的无效劳动是重复劳动，即相似的软件在几个地方同时开发。这种情况多是因项目开发计划不当，或者开发信息不流畅造成的。为此，要建立互相交流、信息往来通畅和具有横向交流特征的信息流通网。

8）提高计划和管理质量

对于大型软件项目来说，提高工程项目管理能力是极其重要的。必须重视对项目开发初期计划阶段的项目计划评价、对计划执行过程中及计划完成报告的评价，将评价、评审工作在工程实施之前就列入整个开发工程的工程计划之中。

4. 基本目标

- 目标1：软件质量保证工作是有计划进行的。
- 目标2：客观地验证软件项目产品和工作是否遵循恰当的标准、步骤和需求。
- 目标3：将软件质量保证工作及结果通知给相关小组和个人。
- 目标4：高级管理层接触到在项目内部不能解决的不符合类问题。

5. 软件开发各个阶段 SQA 的目标

1）需求分析

- 确保客户提出的要求是可行的。
- 确保客户了解自己提出的需求的含义，并且这个需求能够真正达到他们的目标。
- 确保开发人员和客户对于需求没有误解或者误会。
- 确保按照需求实现的软件系统能够满足客户提出的要求。

2）软件规格说明

- 确保规格说明能够完全符合、支持和覆盖前面描述的系统需求。

- 可以采用建立需求跟踪文档和需求实现矩阵的方式。
- 确保规格说明满足系统需求的性能、可维护性、灵活性的要求。
- 确保规格说明是可以测试的,并且制订了测试策略。
- 确保建立了可行的、包含评审活动的开发进度表。
- 确保建立了正式的变更控制流程。

3）设计

- 确保建立了设计的描述标准,并且按照该标准进行设计。
- 确保设计变更被正确地跟踪、控制、文档化。
- 确保设计按照评审准则评审通过并被正式批准之后,才开始正式编码。
- 确保对设计的评审按照进度进行。

4）编码

- 确保建立了编码规范、文档格式标准,并且按照该标准进行编码。
- 确保代码被正确地测试和集成,代码的修改符合变更控制和版本控制流程。
- 确保按照计划的进度编写代码。
- 确保按照进度进行代码评审。

5）测试

- 确保建立了测试计划,并按照测试计划进行测试。
- 确保测试计划覆盖了所有的系统规格说明和系统需求。
- 确保经过测试和调试,软件仍旧符合规格说明和需求定义。

6）维护

- 确保代码和文档同步更新,保持一致。
- 确保建立了变更控制流程和版本控制流程,并按照这些流程管理维护过程中的软件产品变化。
- 确保代码的更改仍旧符合编码规范、通过代码评审,并且不会破坏整个代码结构。

10.3.3　软件质量保证活动

1. 建立SQA小组

20世纪70年代,美国军方在软件开发合同中首次提出了软件质量保证的标准,认为软件质量保证活动的定义是为了保证软件质量而必需的“有计划的和系统化的行动模式”这一观点。该定义的含义是要求在一个组织中应当由多个机构共同协作,承担保证软件质量的责任。包括软件工程师、项目管理者、客户、销售人员和SQA小组的人员。

做技术工作的软件工程师通过采用可靠的技术方法和措施,进行正式的技术评审,执行计划周密的软件测试来考虑质量问题,并完成软件质量保证和质量控制活动。SQA小组是软件开发组织中独立于任何项目组的专职品质保证组织。他们以客户的观点来看待软件,通过自己的工作回答软件是否满足各项质量指标、软件开发是否按照预先设定的标准进行、作为SQA活动一部分的技术规程是否恰当地发挥了作用等问题。SQA小组负责质量保证的计划、监督、记录、分析及报告工作,其职责是辅助软件工程小组得到高质量的最终产品,包括完成以下工作。

（1）为项目准备 SQA 计划。该计划在制定项目、规定项目计划时确定，由所有感兴趣的相关部门评审，其内容包括：

- 需要进行的审计和评审
- 项目可采用的标准
- 错误报告和跟踪的规程
- 由 SQA 小组产生的文档
- 向软件项目组提供的反馈数量

（2）参与开发项目的软件过程描述。评审过程描述以保证该过程与组织政策、内部软件标准、外界标准以及项目计划的其他部分相符。

（3）评审各项软件工程活动，对其是否符合定义好的软件过程进行核实。记录、跟踪与过程的偏差。

（4）审计指定的软件工作产品，对其是否符合事先定义好的需求进行核实。对产品进行评审，识别、记录和跟踪出现的偏差；对是否已经改正进行核实；定期将工作结果向项目管理者报告。

（5）确保软件工作及产品中的偏差已记录在案，并根据预定的规程进行处理。

（6）记录所有不符合的部分并报告给高级领导者。

独立的 SQA 小组是衡量软件开发活动优劣的尺度之一。SQA 小组的独立性，使其享有一项关键权利，即"越级上报"。当 SQA 小组发现产品质量出现危机时，它有权向项目组的上级机构直接报告这一危机。这一形式使许多问题可以在组内得以解决，提高了软件开发的质量和效率。

2. SQA 活动

软件质量保证是 CMM 可重复级中的一个 KPA。软件质量保证的定义为：对软件产品和活动的评审和审计，以验证它们是否符合合适的规程和标准，同时给项目负责人和其他有关负责人提供评审和审计的结果。SQA 小组具体要实施的活动包括：

- 识别质量需求
- 参与项目计划的制定
- 制定 SQA 计划
- 按照 SQA 计划评审工作产品
- 按照 SQA 计划实施审核工作
- 编写 SQA 报告，通知相关人员
- 处理不合格项
- 监控软件产品的质量
- 采集软件质量保证活动的数据
- 度量软件质量保证活动

1）识别质量需求

SQA 小组应参与项目组的需求开发工作，站在用户的角度，协助项目组识别质量指标和可能的质量风险，反映在系统需求中。并且在项目开始时，SQA 小组就应为项目组配备 SQA 人员。

2）参与项目计划的制定

- SQA 小组进行有关项目计划、标准和规程的咨询。
- SQA 小组验证项目计划、标准、规程是否到位，且可用于评审和审核项目。
- SQA 小组参与项目计划的评审。

3）制定 SQA 计划

- 在项目计划制定的同时，SQA 小组负责制定 SQA 计划。
- 项目经理、项目组、SCM 小组评审 SQA 计划。
- SQA 计划经 SQA 经理审核、CCB 批准后被纳入配置管理。

在 SQA 计划中必须明确定义在软件开发的各个阶段是如何进行质量保证活动的，通常包括如下内容：

- 确定质量目标。
- 定义每个开发阶段的开始和结束边界。
- 详细策划要进行的质量保证活动。
- 明确质量活动的职责。
- 明确 SQA 小组的职责和权限。
- 明确 SQA 小组的资源需求，包括人员、工具和设施。
- 定义由 SQA 小组执行的评估。
- 定义由 SQA 小组负责组织的评审。
- SQA 小组进行评审和检查时所参考的项目标准和过程。
- 需要由 SQA 小组创建的文档。

选择合适的 SQA 工具并不是试图通过工具来保证软件产品的质量，而是用以支持 SQA 的活动。选择 SQA 工具时，首先需要明确质量保证目标，根据目标制定选择 SQA 工具的需求并文档化，包括对平台、操作系统及 SQA 工具与软件工程平台接口的要求等。

4）按照 SQA 计划评审工作产品

- 依据 SQA 计划，SQA 小组可以以下列方式评审工作产品：SQA 小组参与项目组评审，SQA 小组独立对工作产品评审，SQA 小组邀请别的专家评审工作产品。
- 依据适用的标准、规程和合同需求，SQA 小组客观地评价工作产品。
- SQA 小组识别和记录工作产品中的不合格项，验证纠正结果，跟踪到问题的解决。

5）按照 SQA 计划实施审核工作

- 根据 SQA 计划，SQA 小组审核项目组和相关的活动，评价其与计划、适用的标准和规程的一致性。
- SQA 小组记录和识别项目活动中的不合格项，验证纠正结果，跟踪到问题的解决。

6）编写 SQA 报告，通知相关人员

- SQA 小组应及时将审核报告或不合格项报告提交给项目经理及项目组相关人员。
- SQA 人员定期（一般是每周）将 SQA 报告提交给项目经理和 SQA 经理。
- SQA 经理定期（一般是每月）将 SQA 报告提交给高层管理和 SEPG。

7）处理不合格项

- SQA 小组将不合格报告提交给项目组相关人员。
- 项目经理负责在规定的期限内进行处理。

- SQA 小组将项目组未能及时处理的不合格项报告高层管理者(事业部、研究所高层管理或产品办公室)。
- 高层管理者负责对这些不合格项进行裁决。
- SQA 人员跟踪不合格项直到它们被改正。

8）监控软件产品的质量

- 对软件产品的验收。
- 把握采购软件的质量。
- 监控分承包商的软件质量保证工作。

9）采集软件质量保证活动的数据

- 记录不协调事项。
- 跟踪不协调事项直到它们被解决。
- 收集各阶段的评审和审计情况。

10）度量软件质量保证活动

度量的目的是为了判断 SQA 活动的成本和进度状态，其内容包括：

- 与其计划相比，SQA 活动完成的里程碑数。
- 在 SQA 活动中完成的工作、花费的工作量及支出的费用。
- 与其计划相比，产品审计和活动评审的次数。

3. SQA 活动的验证

SQA 活动应接受以下验证：

(1）事业部、研究所管理层、产品办公室定期评审或进行由事件驱动的评审 SQA 活动。

(2）项目经理定期评审或进行由事件驱动的评审 SQA 活动。

(3）SEPG 或其他研究所的 SQA 小组定期评审或进行由事件驱动的评审 SQA 活动。

(4）在合适时，客户的 SQA 人员定期评审 SQA 活动。

10.3.4 软件质量保证的措施

软件质量保证的措施主要有基于非执行的测试、基于执行的测试和程序正确性证明。

1. 基于非执行的测试(也称为复审或评审)

软件复审主要用来保证在编码之前各个阶段产生的文档的质量，是软件工程过程中滤除缺陷的“过滤器”，目的是尽可能多地发现被复审对象中的缺陷，起到“净化”工作产品的作用。在软件项目开发过程中的多个不同的点上，软件复审活动能够起到及早发现错误进而引发排错活动的作用。由于人们发现别人生产的产品中的缺陷比发现自己产品的缺陷要容易，所以复审应当在不同的工程师之间进行。任何一次复审都是借助人的差异性来达到目标的活动，它的目标包括：

- 指出一个人或一个小组生产的产品所需要进行的改进。
- 确定被审核产品中不需要或者不希望被改进的部分。
- 得到与未复审时相比更加一致，至少更可预测的技术工作的质量，从而使得技术工作更可管理。

复审的方式很多，包括非正式的复审、正式的同行评审、管理复审等。

正式的技术复审（Formal Technical Review，FTR）是一种由技术工程师进行的软件质量保证活动，其目标是：

- 在软件的任何一种表示形式中发现功能、逻辑或实现上的错误。
- 证实经过复审的软件的确满足需求。
- 保证软件的表示符合预定义的标准。
- 得到一种以一致的方式开发的软件。
- 使项目更加容易管理。因为FTR的进行使大量人员对软件系统中原本并不熟悉的部分更加了解，因此，FTR还起到了提高项目连续性和培训后备人员的作用。

FTR的显著优点是，能够较早发现软件错误，从而可防止错误被传播到软件过程的后续阶段。统计数字表明，在大型软件产品中检测出的错误，60%～70%属于规格说明错误或设计错误，而FTR在发现规格说明错误和设计错误方面的有效性高达75%。由于能够检测出并排除掉大部分这类错误，复审可大大降低后续开发和维护阶段的成本。

FTR实际上是一类复审方式，包括“走查”（Walkthrough）、“审查”（Inspection）、“轮查”（Round Robin Review）以及其他软件小组进行的技术评估。每次的FTR都以会议的形式进行，只有经过适当地计划、控制和相关人员的积极参与，FTR才能获得成功。

1）复审会议的组织

从保证会议效果出发，不论进行什么形式的FTR活动，会议的规模都不宜过大，控制在3～5人较好，每个参会人员都要提前进行准备；会议的时间不宜长，控制在两个小时之内。每次复审的对象应当只是整个软件中的某个较小的特定部分，不要试图一次复查整个设计，而要对每个模块或者一小组模块进行复审走查。

FTR的焦点是某个工作产品，比如一部分需求规约、一个模块的详细设计、一个模块的源代码清单等。负责生产这个产品的人通知“复审责任人”产品已经完成，需要复审。复审责任人对工作产品的完成情况进行评估，当确认已经具备复审条件后，准备产品副本，发放给预定要参加复审的复审者。复审者花1～2小时进行准备，通常在第二天召开复审会议。复审会议由复审责任人主持，由产品生产者和所有的复审者参加，并安排专门的记录员。产品生产者在会议上要“遍历”工作产品并进行讲解，复审者则根据各自的准备提出问题，当发现错误和问题时，记录员将逐一进行记录。

在复审结束时，必须做出复审结论，且只能是下列三种之一：

(1) 工作产品可以不经修改地被接收。

(2) 由于存在严重错误，产品被否决(错误改正后必须重新进行复审)。

(3) 暂时接收工作产品(发现了轻微错误需要改正，但改正后不需要再次评审)。

参与复审的所有人员，都必须在结论上签字以表示他们参加了本次FTR，并同意复审小组的结论。

2）复审报告和记录保存

在FTR期间，一名复审者（记录员）主动记录所有被提出来的问题，在会议结束时对这些问题进行小结，并形成一份“复审问题列表”。此外还要形成一份简单的“复审总结报告”阐明如下问题：

(1) 复审对象是什么；

（2）有哪些人参与复审；

（3）发现了什么、结论是什么。

复审报告是项目历史记录的一部分，可以将其分发给项目负责人和其他感兴趣的复审参与方。复审问题列表有两个作用：首先是标识产品中的问题区域；其次将被用作指导生产者对产品进行改进的“行动条目”。在复审总结报告中，复审问题列表应当作为附件。

SQA人员必须参与复审，一方面观察复审过程的合理性，另一方面将会在今后对问题列表中各个问题的改正情况进行跟踪、检查并通报缺陷修改情况，直到复审被通过或问题被彻底解决。

3）复审指南

不受控制的错误复审比没有复审更加糟糕，所以在进行正式的复审之前必须制定复审指南并将其分发给所有的复审参加者，得到大家的认可后，才能依照指南进行复审。正式技术复审指南的最小集合如下所示：

- 复审对象是产品，而不是产品生产者。复审会议的气氛应当是轻松的和建设性的，不要试图贬低或者羞辱别人。通常，有管理职权的成员不宜作为复审者参加会议。
- 制订并严格遵守议程。FTR会议必须保证按照计划进行，不要离题。
- 鼓励复审者提出问题，但限制争论和辩驳。有争议的问题将被记录在案，以待事后解决。
- 复审是以“发现问题”为宗旨的，问题的解决通常由生产者自己或者在别人的帮助下解决，所以不要试图在FTR会议上解决所有问题。
- 必须设置专门的记录员，做好会议记录。
- 为保证FTR的有实效，坚持要求与会者事先做好准备，提交书面的评审意见，并要限制与会人数，将人数保持在最小的必须值上。
- 组织应当为每类要复审的产品（如各种计划、需求分析、设计、编码、测试用例）建立检查表，帮助复审主持者组织FTR会议，并帮助每个复审者都能够把注意力放在对具体产品来说最为关键的问题上。
- 为FTR分配足够的资源和时间，并且要为复审结果所必然导致的产品修改活动分配时间。
- 所有参与复审的人，都应当具备进行FTR的技能，接受过相关的培训。
- 复审以前所做的复审，总结复审工作经验，不断提高复审水平。

2. 基于执行的测试

基于执行的测试即软件测试，需要在程序编写出来之后进行，它是保证软件质量的最后一道防线。

3. 程序正确性证明

测试只能发现程序错误，但不能证明程序无错。因为测试并没有也不可能包含所有数据，只是选择了一些具有代表性的数据，所以它具有局限性。程序正确性证明是通过数学方法严格验证程序是否与对它的说明完全一致。正确性证明的基本思想是证明程序能完成预定的功能，因此应该提供对程序功能的严格数学说明，然后根据程序代码证明程序确实能实

现它的功能说明。

从 20 世纪 50 年代 Turing 开始研究程序正确性证明，人们陆续提出了许多不同的技术方法。虽然这些技术方法本身很复杂，但是它们的基本原理却比较简单。

如果在程序的若干个点上，设计者可以提出关于程序变量及它们的关系的断言，那么在每一点上的断言都应该永远是真的。假设在程序的 $P_1, P_2, \cdots, P_n$ 等点上的断言分别是 $a(1), a(2), \cdots, a(n)$，其中 $a(1)$ 必须是关于程序输入的断言，$a(n)$ 必须是关于程序输出的断言。

为了证明在点 P_i 和 P_{i+1} 之间的程序语句是正确的，必须证明执行这些语句之后将使断言 $a(i)$ 变成 $a(i+1)$。如果对程序内所有相邻点都能完成上述证明过程，则证明了输入断言加上程序可以导出输出断言。如果输入断言和输出断言是正确的，而且程序确实是可以终止的（不包含死循环），则上述过程就证明了程序的正确性。

人工证明程序正确性，对于评价小程序可能有些价值，但是在证明大型软件的正确性时，不仅工作量太大，更主要的是在证明的过程中很容易包含错误，因此是不实用的。为了实用的目的，必须研究能证明程序正确性的自动系统。

10.4　软件项目管理

10.4.1　软件项目管理概述

软件项目管理是为了使软件项目能够按照预定的成本、进度、质量顺利完成，运用专门的知识、技能、工具和方法，对人员（People）、产品（Product）、过程（Process）和项目（Project）进行分析和管理的活动。

软件项目管理的根本目的是为了让软件项目，尤其是大型项目的整个软件生命周期都能在有效地控制下进行，使软件项目能够按照预期的成本、进度、质量顺利地完成并交付用户使用。而研究软件项目管理是为了从已有的成功或失败的案例中总结出能够指导今后开发工作的通用原则和方法，以避免再次失误。

20 世纪 70 年代中期，美国首次提出软件项目管理的概念。当时美国国防部专门研究了软件开发不能按时提交、预算超支和质量达不到用户要求的原因，结果发现 70％的项目是因为管理不善引起的，而非技术原因。于是软件开发者开始逐渐重视软件开发中的各项管理工作。到了 20 世纪 90 年代中期，软件研发项目管理不善的问题仍然存在，据美国软件工程实施现状的调查，软件研发的情况仍然很难预测，大约只有 10％的项目能够在预定的费用和进度下交付。据统计，1995 年美国共取消了 810 亿美元的商业软件项目，其中 31％的项目未做完就被取消，53％的软件项目进度通常要延长 50％的时间，只有 9％的软件项目能够及时交付并且费用也控制在预算之内。

软件项目管理和其他的项目管理相比有相当的特殊性。首先，软件是纯知识产品，其开发进度和质量很难估计、度量，生产效率也难以预测和保证；其次，软件系统的复杂性也导致了开发过程中各种风险的难以预见和控制。Windows 这样的操作系统有一千五百万行以上的代码，同时有数千个程序员在进行开发，有上百个项目经理，这样庞大的系统如果没有很好的管理，其软件质量是难以想象的。

软件项目管理的主要活动通常包括以下五个方面。

1. 软件生命周期管理

软件生命周期管理就是把整个软件生命周期划分为若干阶段，使得每个阶段有明确的任务，使规模大、结构复杂和管理复杂的软件开发变得容易控制和管理。通常，软件生命周期包括可行性分析、项目启动、需求分析、设计（概要设计和详细设计）、编码、测试、维护等活动，可以将这些活动以适当的方式分配到不同的阶段去完成。

2. 软件项目成本管理

软件项目成本管理，即根据软件开发组织的情况和项目的具体要求，利用组织既定的资源，在保证项目的进度、质量得到客户满意的情况下，对软件项目成本进行有效地组织、实施、控制、跟踪、分析、考核等一系列管理活动，最大限度地降低项目成本，提高项目利润。项目成本管理包括确保在批准的预算范围内完成项目所需要的各个过程，软件项目成本的管理基本上可以用估算和控制来概括，主要内容有以下几个。

- 资源计划：包括决定为实施项目活动需要使用什么资源（人员、设备和物资）以及每种资源的用量，其主要输出是一个资源需求清单。
- 成本估计：估计完成项目所需要的资源成本的近似值。
- 成本预算：将整个成本估算配置到各单项工作，以建立一个衡量绩效的基准计划，其主要输出是成本基准计划。
- 成本控制：控制项目预算的变化，其主要输出修正的成本估算、更新预算、纠正行动和取得的教训。
- 成本预测：项目成本预测是指在项目的实施过程中，依据项目成本的实施发生情况和各种影响因素的发展与变化，不断地预测项目成本的发展和变化趋势与最终可能出现的结果，从而为项目的成本控制提供决策依据的工作。

成本管理计划是成本控制的标准，不合理的计划可能使项目失去控制，超出预算。因此成本估算是整个成本管理过程中的基础，成本控制是使项目的成本在开发过程中控制在预算范围之内。

3. 软件项目时间管理

合理地安排项目时间是项目管理中的一项关键内容，项目时间管理就是采用科学的方法确定目标进度，编制进度计划和资源供应计划，进行进度控制，在与质量、费用目标协调的基础上，实现项目工期目标。其目的是保证按时完成项目、合理分配资源、达到最佳工作效率。它的主要工作包括定义项目活动、任务、活动排序、每项活动的合理工期估算、制定项目完整的进度计划、资源共享分配、监控项目进度等内容。

4. 软件项目人力资源管理

软件项目人力资源管理就是有效地发挥每一个参与项目人员的作用，让项目的所有相关人员能够在可控状态下有条不紊地进行项目的开发活动，包括组织和管理项目团队所需要的所有过程。项目团队由为完成项目而承担了相应的角色和责任的人员组成，团队成员

应该参与大多数项目计划和决策工作。项目团队成员就是项目的人力资源，他们的早期参与能在项目计划过程中增加专家意见和加强项目的沟通。

5. 软件开发质量管理

软件开发质量管理，就是为了开发出符合质量要求的软件产品，贯穿于软件开发生命周期过程的质量管理工作。

10.4.2 软件项目成本管理

1. 基本概念

项目成本管理主要与完成活动所需要的资源成本有关，同时也考虑决策对项目产品使用成本的影响。例如，减少设计方案的次数可减少产品的成本，但却增加了今后顾客的使用成本，这个广义的项目成本被称为项目的生命周期成本。狭义的项目成本（费用）是指因为项目而发生的各种资源耗费的货币体现。成本管理包括项目资源规划、成本估算、成本预算、成本控制和成本预测。

1）成本

成本就是为了获取商品或服务而支付的货币总量。软件项目的成本，就是为了使软件项目如期完成，而支付的所有费用。软件项目成本可以从以下两个方面来分析：成本与质量、时间的关系；在预算框架内控制成本。

2）规模

软件项目规模即工作量，是从软件项目范围中抽出的软件功能，然后确定每个软件功能所必须执行的一系列软件工程任务。包括软件规划、软件管理、需求、设计、编码、测试以及后期的维护等任务。软件成本包括完成软件规模相应付出的代价，待开发的软件项目需要的资金，其中人的劳动消耗所需要的代价是软件产品的主要成本。

规模是成本的主要因素，是成本估算的基础，有了规模就确定了成本。

3）成本管理

项目成本管理是指为保障项目实际发生的成本不超过项目预算，使项目在批准的预算内按时、按质、经济高效地完成既定目标而开展的成本管理活动。

4）成本构成

从软件生命周期构成的两个阶段即开发阶段和维护阶段看，软件的成本由开发成本和维护成本构成。其中开发成本由软件开发成本、硬件成本和其他成本组成，包括了系统软件的分析或设计费用（包含系统调研、需求分析、系统设计）、实施费用（包含编程或测试、硬件购买与安装、系统软件购置、数据收集、人员培训）及系统切换等方面的费用；维护成本由运行费用（包含人工费、材料费、固定资产折旧费、专有技术及技术资料购置费）、管理费（包含审计费、系统服务费、行政管理费）及维护费（包含纠错性维护费用及适应性维护费用）组成。

从财务角度来看，软件项目的成本如下所示。

- 硬件购置费：例如，计算机及相关设备的购置，不间断电源、空调等的购置费。
- 软件购置费：例如，操作系统软件、数据库系统软件和其他应用软件的购置费。
- 人工费：主要是开发人员、操作人员、管理人员的工资福利费等。

- 培训费。
- 通信费：例如，购置网络设备、通信线路器材、租用公用通信线路等的费用。
- 基本建设费：例如，新建、扩建机房、购置计算机机台、机柜等的费用。
- 财务费用。
- 管理费用：例如，办公费、差旅费、会议费、交通费。
- 材料费：如打印纸、包带、磁盘等的购置费。
- 水、电、汽费。
- 专有技术购置费。
- 其他费用：例如，资料费、固定资产折旧费及咨询费。

5）成本的影响因素

（1）项目质量对成本的影响。

一个项目的实现过程就是项目质量的形成过程，在这一过程中为达到质量要求需要开展两个方面的工作：质量的检验与保障工作和质量失败的补救工作。这两项工作都要消耗资源，从而都会产生项目的质量成本。质量与费用之间的关系如图 10-18 所示。

（2）工期对成本的影响。

项目的工期是整个项目、项目某个阶段，或某项具体活动所需要或实际花费的工作时间周期。工期对成本的影响如图 10-19 所示。

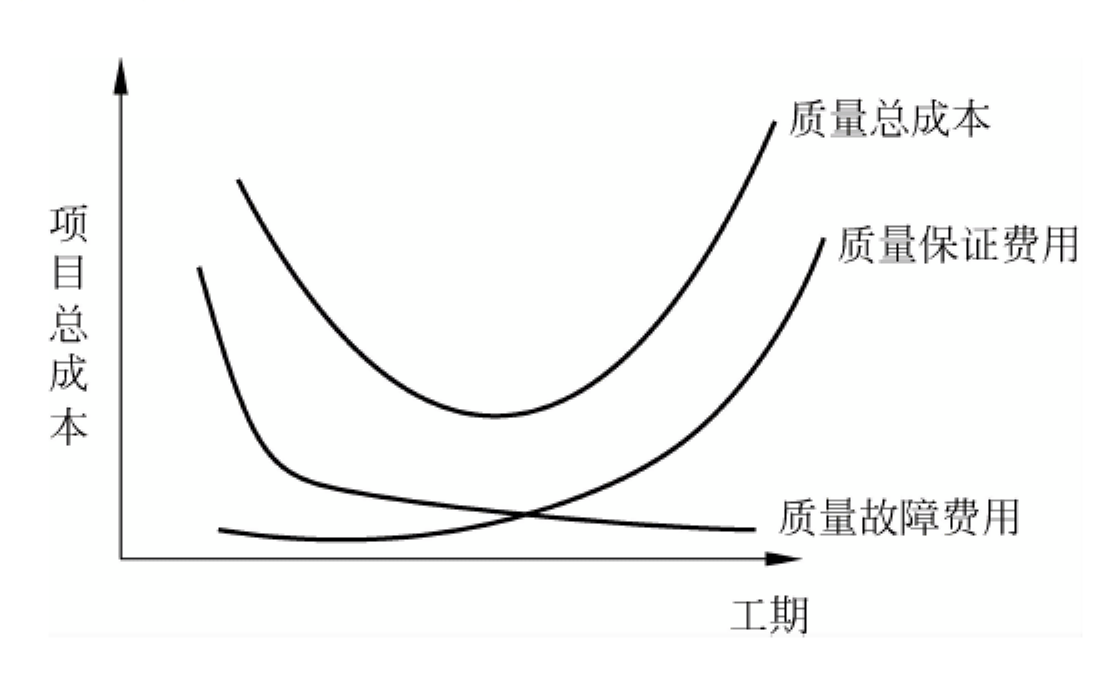


图 10-18 质量与费用之间的关系

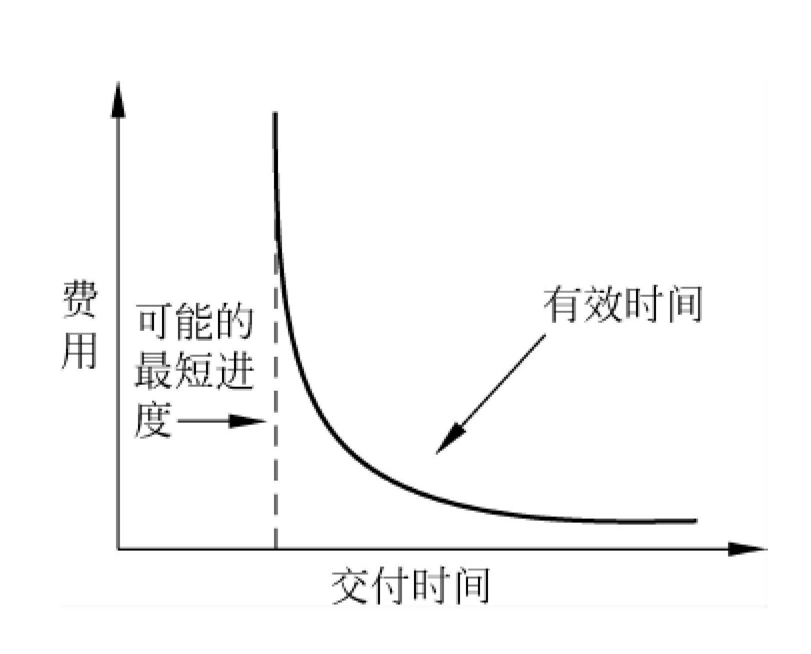


图 10-19 工期对成本的影响

（3）管理水平对成本的影响。

- 项目成本预算和估算的准确度差。由于客户需求的不断变化，项目工作内容和工作量也随之不断改变。一旦发生变化，项目经理就追加项目预算，使得预算频频变更，等到项目结束时，实际成本和初始计划偏离很大。此外，项目预算往往会走两个极端：过粗和过细。预算过粗会使项目费用的随意性较大，准确度降低；预算过细会使项目控制的内容过多，弹性差、变化不灵活、管理成本加大。
- 缺乏对软件成本事先估计的有效控制。在开发初期，对成本不够关心，忽略对成本的控制，只有在项目进行到后期，实际远离计划出现偏差的时候，才进行成本控制，这样往往导致项目超出预算。
- 缺乏成本绩效的分析和跟踪。在传统的项目成本管理中，将预算和实际数值对比，但很少有将预算、实际成本和工作量进度联系起来，考虑实际成本和工作量是否匹配的问题。

（4）人力资源对成本的影响。

人力资源素质也是影响成本的重要因素。对高技术能力、高技术素质的人才，本身的人力资源成本是比较高的，但可以产生高的工作效率、高质量的产品、较短的工期等间接效果，从而总体上会降低成本；而对于一般人员，还需要技术培训，对项目的理解及工作效率相对低下，工期会延长，需要雇佣更多的人员，造成成本的增加。因此，人力资源也是重要的影响因素。

（5）价格对成本的影响。

中间产品和服务、硬件、软件的价格也对成本产生直接的影响，价格对项目预算的估计影响很大。

2. 资源计划

可将资源理解为一切具有现实和潜在价值的东西，完成项目必须要消耗的劳动力（人力资源）、材料、设备、资金等有形资源，同时还可能需要消耗其他一些无形资源，而且由于存在资源约束，项目耗用资源的质量、数量、均衡状况对项目的工期、成本有着不可估量的影响。项目资源计划是指通过分析、识别和确定项目所需资源种类（人力、设备、材料、资金等）、数量和投入时间，制定科学、合理、可行的项目资源供应计划的项目成本管理活动。在项目资源计划工作中最为重要的是确定出能够充分保证项目实施所需各种资源的清单和资源投入的计划安排。

1）资源计划的主要依据

- 工作分解结构（Work Breakdown Structure，WBS）。
- 项目进度计划。
- 历史资料。
- 资源库描述：对项目拥有的资源存量的说明。
- 组织策略：项目实施组织的企业文化和组织结构，项目组织获得资源的方式和手段方面的方针，均体现了项目高层在资源使用方面的策略，可以影响到人员招聘、物资和设备的租赁或采购，对如何使用资源起重要作用。

2）资源计划的编制步骤

- 资源需求分析：包括工作量计算，确定实施方案，估计人员需求量，估计设备、材料需求量，确定资源的使用时间。
- 资源供给分析。
- 资源成本比较与资源组合。
- 资源分配与计划编制。

3）编制资源计划的方法与工具

（1）德尔菲（专家）评估法。

由项目成本管理专家根据经验和判断去确定和编制项目资源计划的方法。这种方法通常有两种具体的形式：专家小组法与德尔菲法。

德尔菲法的具体做法是：

- 设计调查表
- 选择应答的专家

- 征询专家的意见

德尔菲法具有经济性、匿名性和客观性的特点，且周期较短、费用较低。但是该方法对各种意见的可靠程度和科学依据缺乏统一的标准，理论上缺乏深刻的逻辑论证。

(2) 资料统计法。

使用历史项目的统计数据资料，计算和确定项目资源计划的方法。

- 实物量指标多数用来表明物质资源的需求数量，这类指标一般表现为绝对数指标。
- 劳动量指标主要用于表明人力的使用，这类指标可以是绝对量也可以相对量指标。
- 价值量指标主要用于表示资源的货币价值，一般使用本国货币币值表示活劳动或物化劳动的价值。

常用的项目资源计划的工具包括资源矩阵(见表 10-3)、资源数据表(见表 10-4)、资源甘特图、资源负载图或资源需求曲线(见图 10-20)、资源累计需求曲线(见图 10-21)等。

表 10-3 资源矩阵

工 作	资源需要					相关说明
工作 1 工作 2 ⋮ 工作 m	资源 1	资源 2	…	资源 3	资源 n	

表 10-4 资源数据表

资源需求种类	资源需求总量	时间安排(不同时间资源需求量)						相关说明
		1	2	3	…	$T-1$	T	
资源 1 资源 2 ⋮ 资源 n								

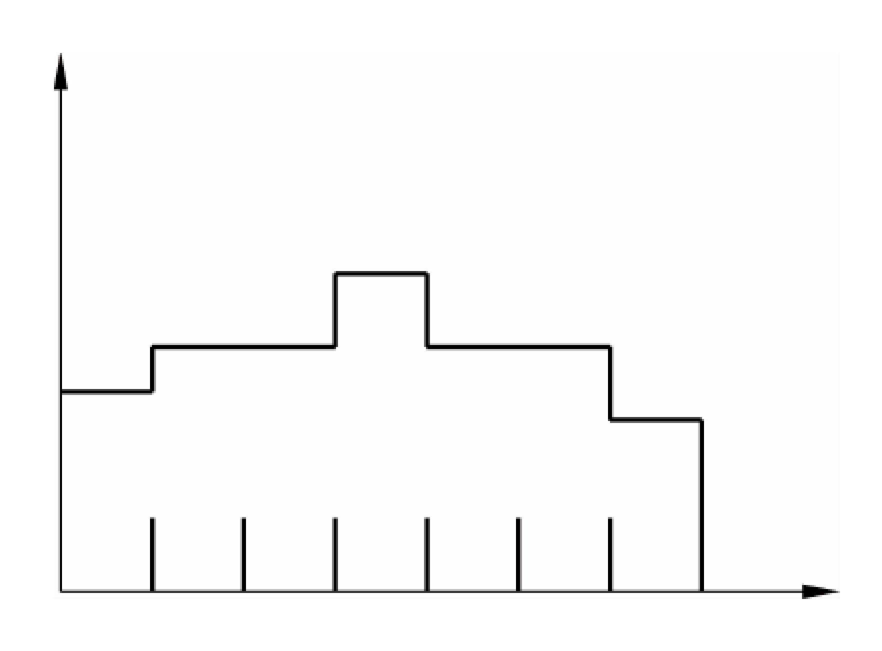


图 10-20 资源甘特图、资源负载图或资源需求曲线

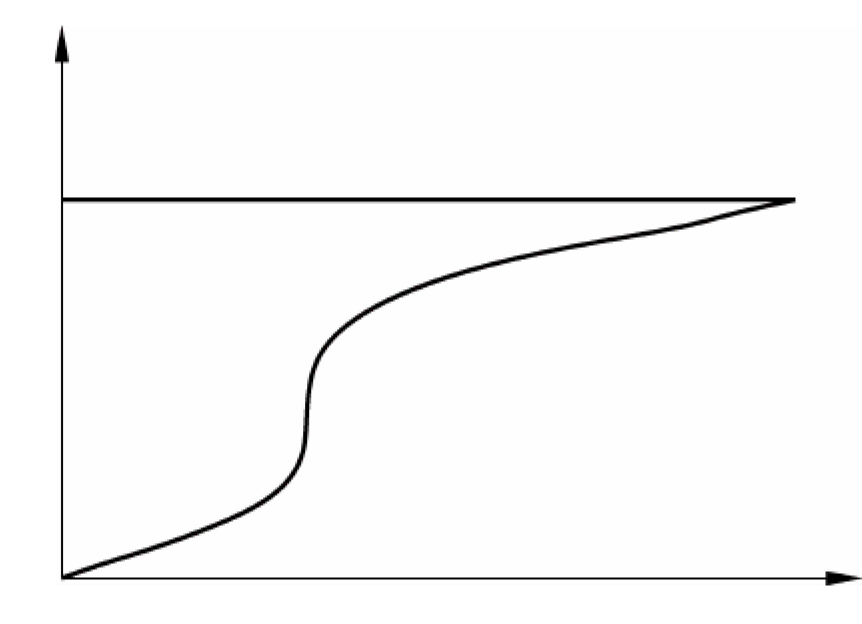


图 10-21 资源累计需求曲线

3. 成本估计

成本估计是对完成项目各项任务所需要资源的成本所进行的近似估算，主要依据包括：

- 项目范围说明
- WBS

- 资源计划
- 资源消耗率
- 历史信息(同类项目的历史资料始终是项目执行过程中可以参考的最有价值的资料,包括项目文件、共用的费用、估算数据及项目工作组的知识等)

1) 成本估计过程

成本估计过程如图 10-22 所示,在进行软件成本估算时,还应该更多地考虑以下一些影响因素:需求的不确定性、计划的不落实性、规模和工作量的不确定性、人员属性以及外部环境等影响因素对成本估计的影响。

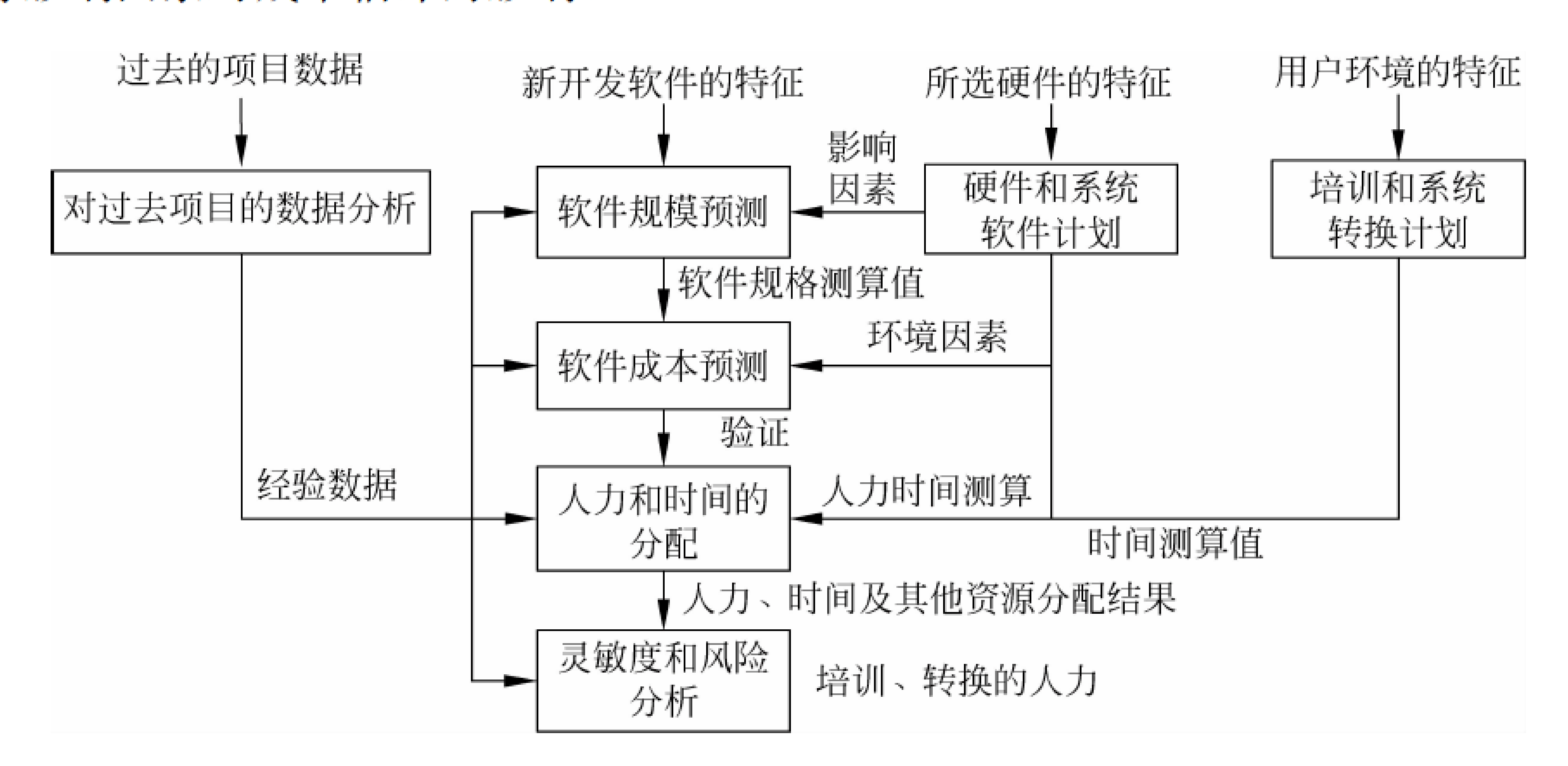


图 10-22 成本估计过程

2) 成本估计的技术路线

(1) 自顶向下地估计:就是根据项目管理人员的经验和判断,结合以前相关类似活动的历史数据,上层管理人员估计项目整体的成本和子项目的成本,然后把这个估计的成本给下一层的管理人员,该层的层管理人员再对任务和子任务的成本进行估计,直到最底层,如图 10-23 所示。自顶向下的估计通常在项目的初期或信息不足时进行,此时只确定了初步的 WBS,分解层次少,估算精度较差。这种成本估计实际上是以项目成本总体为估算对象,在收集上层和中层管理人员的经验判断,以及可以获得的关于以往类似项目历史数据的基础上,将成本从工作分解结构的上部向下部依次分配、传递,直至 WBS 的最底层。

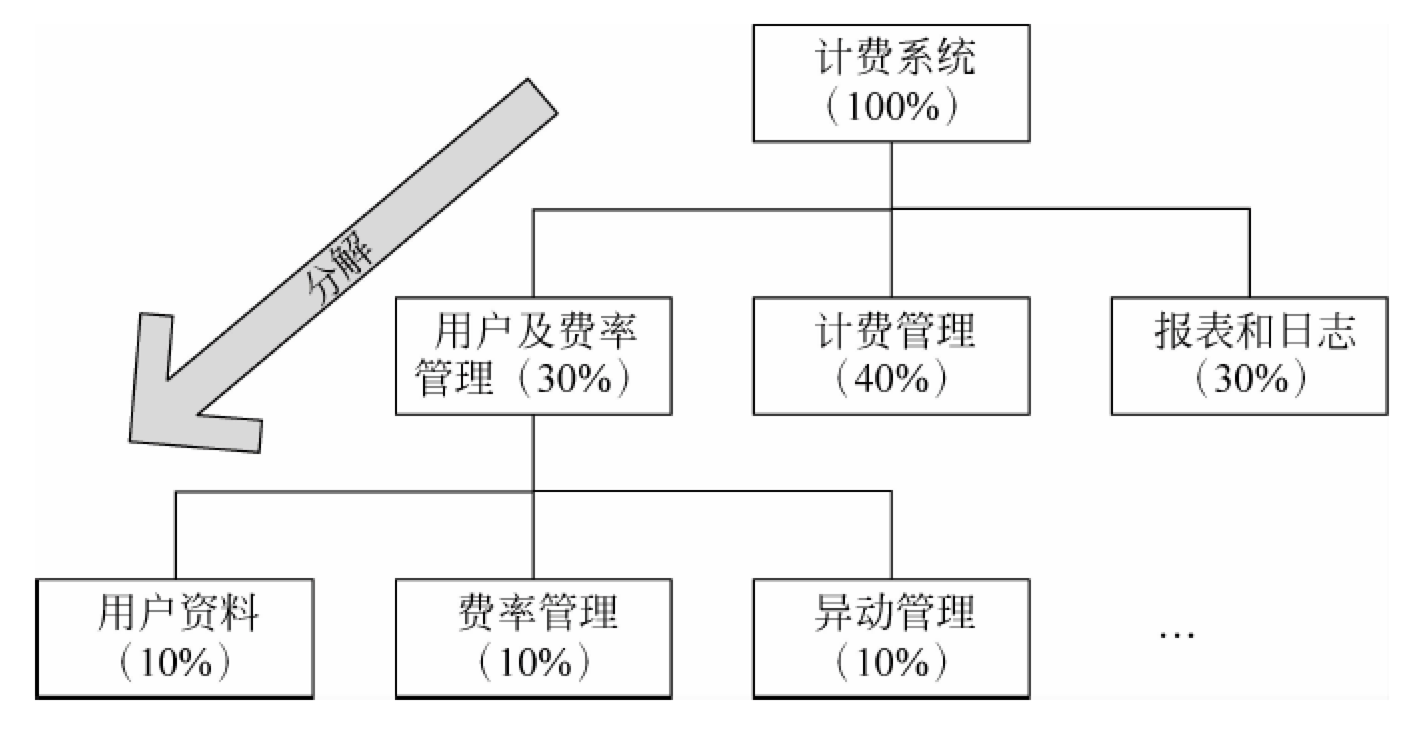


图 10-23 某计费系统项目成本的自顶向下估计

（2）自底向上地估计：自底向上地估计是先估算各个工作包的费用，然后自下而上将所有估算结果汇总，算出项目费用总和。进行自底向上地估计时，必须对任务的时间和资源进行确定，然后把资源转换为所需要的经费，如图 10-24 所示。这个转换有时候需要进行某些修正，而且要和管理层一致。如果不一致，则需要进行协商和沟通来保证估算的精度。所有任务的估算的总和再加上间接成本（如管理成本等）就是项目完成所需要的估算值。采用这种技术路线的前提是确定了详细的 WBS，能做出较准确的估算。当然，这种估算本身要花费较多的费用。

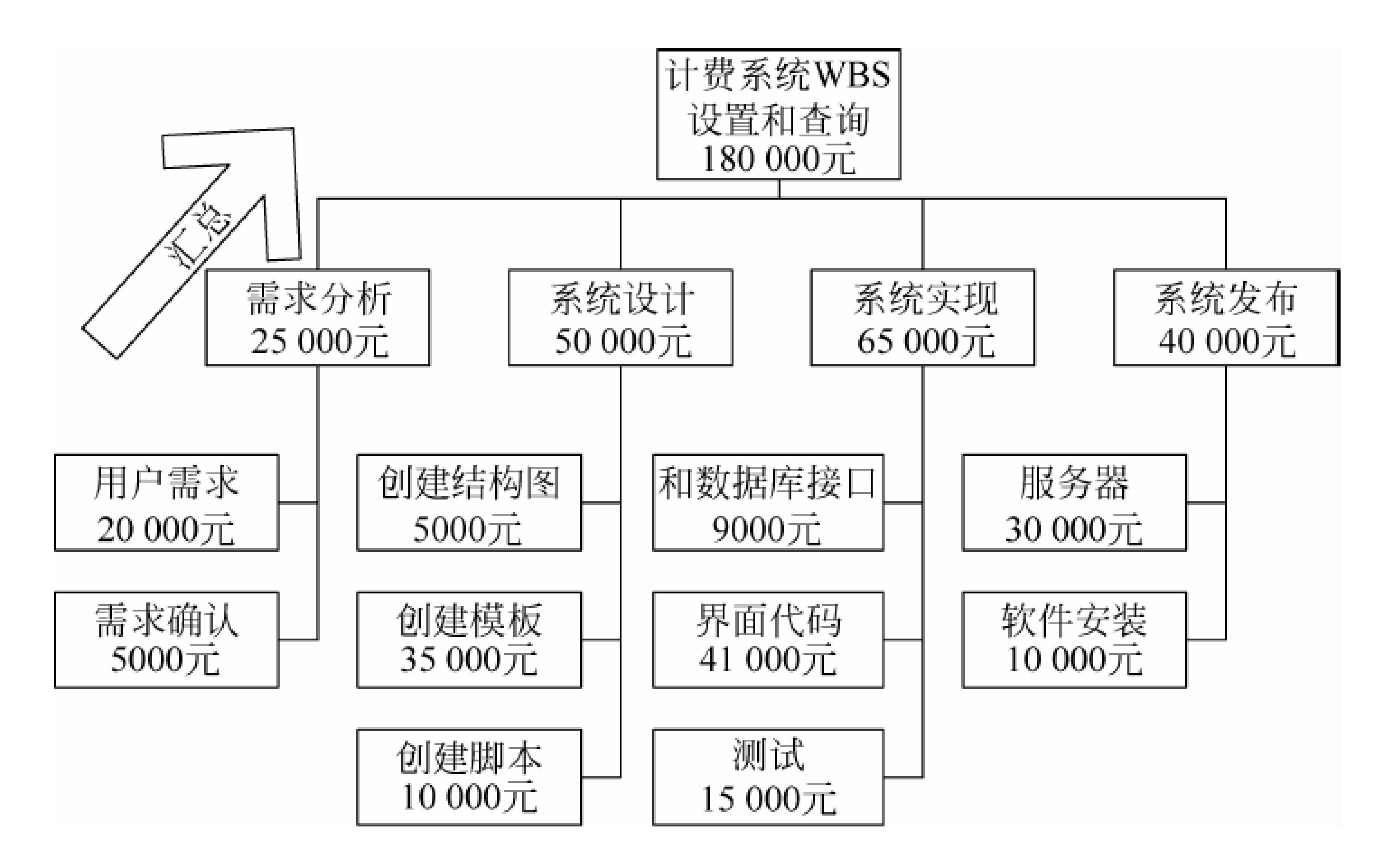


图 10-24 计费系统成本自底向上估算

（3）自顶向下和自底向上相结合的成本估算。

3）成本估算方法

除了代码行技术和功能点技术之外，在 WBS 基础上进行全面详细估算也是软件开发成本估算的常用方法。

利用 WBS 方法，先把项目任务进行合理地细分，分到可以确认的程度，如某种材料、某种设备、某一活动单元等。然后估算每个 WBS 要素的费用。采用这一方法的前提条件如下所示。

（1）对项目需求做出一个完整的限定。

对项目需求的完整限定应包括工作报告书、规格书以及总进度表。工作报告书是指实施项目所需的各项工作的叙述性说明，它应确认必须达到的目标。如果有资金等限制，该信息也应包括在内。规格书是对工时、设备以及材料标价的根据，它能使项目人员和用户了解估价的依据。总进度表应明确项目实施的主要阶段和分界点，其中包括长期订货、原型试验、设计评审会议以及其他任何关键的决策点。如果可能，用来指导成本估算的总进度表应含有项目开始和结束的日历时间。

（2）制定完成任务所必需的逻辑步骤。

一旦项目需求被勾画出来，就应制定完成任务所必需的逻辑步骤。在现代大型复杂项目中，通常是用箭线图来表明项目任务的逻辑程序，并以此作为下一步绘制 CPM（Critical Path Method）或 PERT（Program Evaluation And Review Technique）图以及 WBS 表的根据。

（3）编制 WBS 表。

编制 WBS 表的最简单方法是依据箭线图，把箭线图上的每一项活动当作一项工作任

务，在此基础上再描绘分工作任务。

完成进度表和 WBS 表之后，就可以进行成本估算了。在大型项目中，成本估算的结果最后应以下述的报告形式表述出来。

- 对每个 WBS 要素的详细费用估算。还应有一个各项分工作、分任务的费用汇总表，以及项目和整个计划的累积报表。
- 每个部门的计划工时曲线。如果部门工时曲线含有“峰”和“谷”，应考虑对进度表做若干改变，以得到工时的均衡性。
- 逐月的工时费用总结。以便项目费用必须被削减时，项目负责人能够利用此表和工时曲线做权衡性研究。
- 逐年费用分配表。此表以 WBS 要素来划分，表明每年（或每季度）所需费用，实质上是每项活动的项目现金流量的总结。

采用这种方法估算成本需要进行大量的计算，工作量较大，所以仅计算本身就需要花费一定的时间和费用。但这种方法的准确度较高，用这种方法做出的上述报表不仅仅是成本估算的表述，还可以用来作为项目控制的依据。最高管理层则可以用这些报表来选择和批准项目，评定项目的优先性。

4）经验成本估算模型

（1）面向 KLOC 的估算模型。

- Walston_Felix 模型：$E = 5.2\times(KLOC)0.91$
- Bailey_Basili 模型：$E = 5.5+0.73\times(KLOC)1.16$
- Boehm 简单模型：$E = 3.2\times(KLOC)1.05$
- Doty（在 KLOC>9 时适用）：$E = 5.288\times(KLOC)1.047$

（2）面向功能点（Function Point，FP）的估算模型。

- Albrecht & Gafffney 模型：$E = -13.99+0.0545FP$
- Maston，Barnett 和 Mellichamp 模型：$E = 585.7+15.12FP$

从（1）和（2）列出的模型可以看出，对于相同的 KLOC 或 FP 值，用不同模型估算将得出不同的结果。主要原因是，这些模型多数都是仅根据若干应用领域中有限个项目的经验数据推导出来的，适用范围有限。因此，必须根据当前项目的特点选择适用的估算模型，并且根据需要适当地调整（例如，修改模型常数）估算模型。

（3）SLIM 模型。

1979 年前后，Putnam 在美国计算机系统指挥中心资助下，对 50 个较大规模的软件系统花费估算进行研究，并提出 SLIM 商业化的成本估算模型，SLIM 基本估算方程（又称为动态变量模型）式为：

$$L = C_K K^{\frac{1}{3}} {t_d}^{\frac{4}{3}} \tag{10-1}$$

式中，L 和 t_d 分别表示可交付的源指令数和开发时间（单位为年）；K 是整个生命周期内人的工作量（单位为人年），可从总的开发工作量 ED=0.4K 求得；C_K 是根据经验数据而确定的常数，表示开发技术的先进性级别。在较差的开发环境中（没有一定的开发方法、缺少文档、评审或批处理方式），取 $C_K=6500$；在正常的开发环境（有适当的开发方法，较好的文档和评审，以及交互式的执行方式）中，$C_K=10\ 000$；在较好的开发环境（自动工具和技术）中，则取 $C_K=12\ 500$。

变换式(10-1),可得开发工作量方程为:

$$K=\frac{L^3}{C_K{}^3 t_d{}^4}$$

SLIM除了提供开发时间和成本估算外,还提供可行性、估算CUP时间需求及项目计划中的其他有关信息。

(4) COCOMO模型。

基本COCOMO模型是一个静态单变量模型,它用一个已估算出的源代码行数(LOC)为自变量的函数来计算软件开发工作量。中级COCOMO模型则在用LOC为自变量的函数计算软件开发工作量的基础上,再用涉及产品、硬件、人员、项目等方面属性的影响因素来调整工作量的估算。高级COCOMO模型包括中级COCOMO模型的所有特性,但用上述各种影响因素调整工作量估算时,还要考虑对项目过程中分析、设计等各步骤的影响。模型的核心是方程:

$$\mathrm{ED}=rS^c \text{ 和 } \mathrm{TD}=a(\mathrm{ED})^b \tag{10-2}$$

式中,ED为总的开发工作量(到交付为止),单位为人月;S为源指令数(不包括注释,但包括数据说明、公式或类似的语句);常数r和c为校正因子。若S的单位为10^3,ED的单位为人月。TD为开发时间,经验常数r、c、a和b取决于项目的总体类型(结构型、半独立型或嵌入型),如表10-5所示。

表10-5 项目总体类型

特 性	结 构 型	半独立型	嵌 入 型
对开发产品目标的了解	充分	很多	一般
对软件系统有关的工作经验	广泛	很多	中等
为软件一致性需要预先建立的需求	基本	很多	完全
为软件一致性需要外部接口规格的说明	基本	很多	完全
关联的新硬件和操作过程的并行开发	少量	中等	广泛
对改进数据处理体系结构算法的要求	极少	少量	很多
早期实施费用	极少	中等	较高
产品规模(交付的源指令数)	少于五万行	少于三十万行	任意
实例	• 批数据处理 • 科学模块 • 事务模块 • 熟悉的操作系统 • 编译程序 • 简单的编目生产控制	• 大型事务处理系统 • 新的操作系统数据库管理系统 • 大型编目生产控制 • 简单的指挥系统	• 大而复杂的事务处理系统 • 大型的操作系统 • 宇航控制系统 • 大型指挥系统

通过引入与15个成本因素有关的 r 作用系数将中级模型进一步细化，这15个成本因素如表10-6所示。根据各种成本因素将得到不同的系数，虽然中级COCOMO方程与基本COCOMO方程相同，但系数不同，由此得出中级COCOMO估算方程，如表10-7所示。

表10-6　影响 r 值的15个成本因素

类　型	成本因素
产品属性	①要求的软件可靠性；②数据库规模；③产品复杂性
计算机属性	①执行时间约束；②主存限制；③虚拟机变动性；④计算机周转时间
人员属性	①分析人员能力；②应用经验；③程序设计人员能力；④虚拟机经验；⑤程序设计语言经验
工程属性	①最新程序设计实践；②软件开发工具的作用；③开发进度限制

表10-7　工作量和进度的基本COCOMO方程

开发类型	工　作　量	进　　度
结构型	$ED=2.4S^{1.05}$	$TD=2.5(ED)^{0.38}$
半独立型	$ED=3.0S^{1.12}$	$TD=2.5(ED)^{0.35}$
嵌入型	$ED=3.6S^{1.20}$	$TD=2.5(ED)^{0.32}$

其中经验常数 r、c、a、b 取决于项目的总体类型。

高级COCOMO模型允许将项目分解为一系列的子系统或者子模型，这样可以在一组子模型的基础上更加精确地调整一个模型的属性。当成本和进度的估算过程转换到开发的详细阶段时，就可以使用这一机制。高级的COCOMO对于生命周期的各个阶段使用不同的工作量系数。

5）成本估计结果

（1）成本估计文件。

成本估计文件是对完成软件项目所需要费用的估计和计划安排，对完成项目活动所需要资源、资源成本和数量进行概略或详细地说明。这包括对于项目所需要的人员、设备和其他科目成本估算的全面描述和说明。另外，这一文件还要全面说明和描述项目的不可预见费用等内容。成本估计文件中的主要指标是价值量指标，为了便于在项目实施期间或项目实施后进行对照，该文件也需要使用其他的一些数量指标对成本进行描述。

（2）细节说明文件。

细节说明文件的内容主要包括以下几个。

- 软件项目范围的描述。
- 软件成本估计的基础和依据文件，包括制定项目成本估算的各种依据性文件，各种成本计算或估算的方法说明，以及各种参照的国家规定等。
- 成本估计各种假定条件的说明文件，包括在项目成本估计中所假定的各种项目实施的效率、项目所需资源的价格水平、项目资源消耗的定额估计等假设条件的说明。
- 成本估计可能出现的变动范围的说明。

（3）成本管理计划。

成本管理计划包括管理和控制软件项目成本变动的所有说明文件。软件项目成本管理的核心内容就是这种计划和安排，以及有关项目不可预见费用的使用管理规定。

4. 成本预算

成本估计的输出结果是成本预算的基础与依据。成本预算就是在软件项目成本估计的基础上,更精确地估算项目总成本,并将其分摊到项目的各项具体活动和各个具体项目阶段上,为项目成本控制制定基准计划的成本管理活动,又被称为项目成本计划。

1) 预算的特征

- 计划性。项目预算是一种分配资源的计划,通过既定资源分配,确定项目中各个部分的关系和重要程度,以及对项目中各项活动的支持力度。在项目计划中,项目根据 WBS 被分解为多个工作包,形成一种系统结构,成本预算就是将成本估计总费用尽量精确地分配到 WBS 的每一个组成部分,从而形成与 WBS 相同的系统结构。因此,预算是另一种形式的项目计划。
- 约束性。因为项目高级管理人员在制定预算的时候均希望能够尽可能"正确"地为相关活动确定预算,既不过分慷慨,以避免浪费和管理松散,也不过于吝啬,以免项目任务无法完成或者质量低下,所以成本预算是一种分配资源的计划。成本预算的结果可能并不能满足所有相关人员的利益要求,从而表现为一种约束,所涉及人员只能在这种约束的范围内行动。
- 控制性。成本预算是一种项目成本控制机制,可以作为一种比较标准来使用,是一种度量资源实际使用量和计划用量之间差异的基线标准。由于环境的不确定性,项目预算可能会发生一定的偏离,应尽量将项目的实施与预算的偏差控制在最小的范围之内,所以项目预算的实质就是一种控制机制。

2) 成本预算的核心目标

预算的核心目标就是为了保证整体项目的顺利完成。成本预算过程必须将资源使用情况与组织目标的实现紧密联系起来,否则计划或控制过程就会失去其本来的意义。预算应该以实现最终项目目标为基础,否则,项目管理人员就会忽视最终目标,资金在工作完成之前就被耗用殆尽。

3) 成本预算的依据

项目成本预算的依据有:成本估算文件、WBS、项目进度计划等。

4) 成本预算的编制

(1) 成本预算的编制原则。

为了使成本预算能够发挥其积极作用,在编制成本预算时应遵循以下原则。

- 成本预算要与项目目标相联系,包括项目质量目标和进度目标。成本与质量、进度之间关系密切,三者之间既统一又对立,所以,在通过成本预算确定成本目标时,必须同时考虑到项目质量目标和进度目标。项目质量目标要求越高,成本预算也就越高;项目进度越快,项目成本也会越高。
- 成本预算要以项目需求为基础。项目成本预算与项目需求直接相关,项目需求是项目成本预算的基石。如果以非常模糊的项目需求为基础进行预算,则成本预算没有现实性,容易发生成本超支。
- 成本预算要切实可行。编制成本预算过低,再努力也难以完成项目;实际费用很低,预算过高,便失去作为成本控制基准的意义。故编制成本预算,要根据有关的财经法律、方针政策,从项目的实际情况出发,充分挖掘项目组织的内部潜力,使成本

指标既积极可靠,又切实可行。

- 成本预算应当具有一定的弹性。

(2) 成本预算的编制步骤。

- 分摊总预算成本。
- 制定累计预算成本。

(3) 成本预算的编制过程。

- 单位工程预算编制：常用的方法有单价扩大法、造价指标法、类比预算法。
- 工程项目综合预算编制：综合预算书一般包括编制说明、综合预算表等。
- 工程项目综合预算书的组成。

5) 成本预算的制订方法

(1) 自上而下地进行预算。

自上而下地进行预算需要组织高级管理层的直接输入,实际上,这种方法需要确认高级管理层对成本管理的意见和经验。假设高级管理层具有以往项目的丰富经验,他们不仅能提供精确的反馈,还能为将来的项目风险进行正确地估算。

(2) 自下而上地进行预算。

自下而上地进行预算可汇总 WBS 中各项具体活动的成本,形成项目活动的直接成本和间接成本。这种预算方法的程序是首先将各个工作包的成本相加,形成可交付任务,再将每个任务的预算汇总,形成更高一级的工作项估算。这样把所有活动的总成本相加,最终完成整个项目的总体成本预算。

6) 基于活动的成本预算

基于活动的成本预算工作包括下述四个步骤：

(1) 识别消耗资源的活动,将成本分配给这些活动。

(2) 识别与各个活动相关的成本驱动因素。

(3) 计算每单位成本驱动因素的成本率。

(4) 将成本率与成本驱动因素的单位数量相乘,把成本分配给各个项目。

7) 成本预算结果

(1) 成本基线。

成本基线是成本预算的成果之一,是项目从开始到结束的整个生命周期内的成本累计曲线,以时段估计成本进一步精确、细化编制而成,通常以时间-成本累计曲线(S 曲线)的形式表示,是按时间分段的项目成本预算。成本基线描述了项目生命周期中到某个时间点为止的累计成本支出,是项目管理计划的重要组成部分,用来度量项目的绩效。原始的成本预算就是成本基线,也就是项目的期望成本。成本基线图如图 10-25 所示。

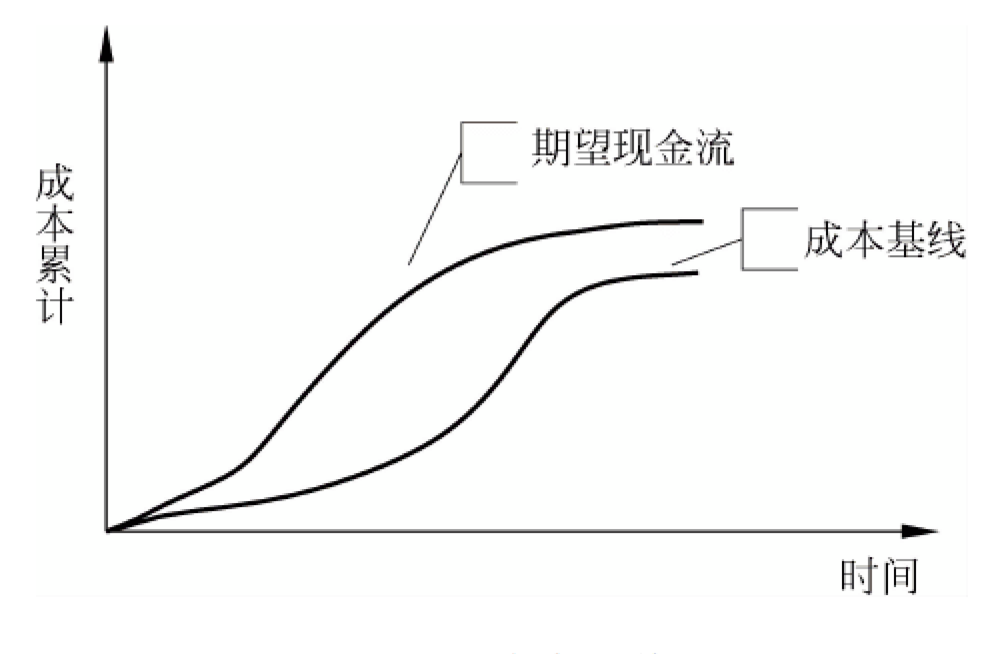


图 10-25 成本基线图

（2）成本预算表(见表 10-8)。

表 10-8 成本预算表

	活动名称	需要的人力资源	需要的时间	单位时间的工资标准(元/天)	小计(元)
人工成本	对现有部分中小型企业的人事软件进行市场调研	项目经理 小组调查人员 10 人	3 天	项目经理 600 调查人员 50	3300
	对调查数据进行分析	项目经理 分析人员 5 人	2 天	项目经理 600 分析人员 500	6200
	调查数据汇总	项目经理 分析人员 5 人	2 天	项目经理 600 分析人员 500	6200
	对需求进行分析并设计出系统	项目经理 设计员 5 人	8 天	项目经理 600 设计员 500	24 800
	领导对需求分析及系统设计提出意见	项目经理	2 天	项目经理 600	1200
	最终的需求分析及系统设计	项目经理 设计员 5 人	5 天	项目经理 600 设计员 500	15 500
	项目经理分配工作对代码及数据库进行实现	项目经理 程序员 3 人 数据库开发人员 3 人	20 天	项目经理 600 程序员 500 数据库开发人员 450	69 000
	测试级完善	测试人员 2 人 程序员 3 人 数据库开发人员 3 人	14 天	测试人员 450 程序员 400 数据库开发人员 450	48 300
	美工处理	美工人员 2 人	9 天	美工人员 400	7200
	最终测试及完善	测试人员 2 人 程序员 3 人 数据库开发人员 3 人 美工人员 2 人	3 天	测试人员 450 程序员 400 数据库开发人员 450 美工人员 400	12 750
	软件交付准备	项目经理	4 天	项目经理 600	2400
	验收	项目经理	2 天	项目经理 600	1200
	系统部署	软件部署人员 2 人	2 天	部署人员 350	1400
	项目结项	全体人员	2 天		
	软件维护	软件维护人员		2000	
	人工成本合计				199 450
非人工成本	非人力资源的分类		非人力资源的数量	单价(元)	小计(元)
	材料	传单	10 000	0.05	500
		相关书籍	20	50	1000
		网络下载所需	50	5	250
		材料合计			1750
	设备	电脑	18 * 44	50	39 600
		投影设备	1 * 44	100	4400
	差旅及通信费	差旅费	2 * 44	40	3520
		电话费	18 * 44	20	15 840
		餐费	18 * 44	20	15 840
		餐饮费	5	500	2500
		差旅及通信费合计			37 700
	管理费用	写字楼租赁费	4	10000	40 000
	其他	茶水费	44 * 5	15	3300
		加班费	10	100	1000
	非人工成本合计				127 750
不可预见费用					7700
总　　计					334 900

5. 成本控制

在发现造成成本偏差的原因后，必须采取相应的措施，以减少成本偏差，把成本控制在计划的范围内，保证目标成本的实现或者修改目标成本。

成本控制一般考虑两种活动，一种是当前正在进行的活动。如果出现了成本偏差，项目管理者不能指望后面的活动会自动减少成本以减少成本偏差。纠正措施越晚，则纠正的可能性就越小，项目成本偏差就可能越来越大。

1）成本失控的主要原因

可能导致项目成本失控的原因有很多，主要原因通常有以下几个：

- 缺乏计划
- 目标不明
- 范围蔓延
- 缺乏领导力

2）成本再预测

项目出现成本偏差，意味着原来的成本预算出现了问题，已完成工作的预算成本和实际成本不相符。这必然会对项目的总体实际成本带来影响，这时候需要重新估算项目的成本。这个被重新估算的成本也被称预测项目未来完工成本（Estimate At Completion，EAC）。EAC 的计算方法有三种：

（1）第一种是认为项目日后的工作将和以前的工作效率相同，未完成工作的实际成本和未完成工作预算的比例与已完成工作的实际成本（Actual Cost of Work Performed，ACWP）和已完成工作的预算（Budgeted Cost of Work Performed，BCWP）的比例相同。

$$EAC=(ACWP/BCWP)\times BAC$$

其中，BAC（Budget At Completion）是项目完工预算，指编写计划时预计的项目完工费用。EAC 是预测的项目完工估算，指计划执行过程中根据当前的进度、费用偏差情况预测的项目完工总费用。

（2）第二种是假定未完成工作的效率和已完成工作的效率没有什么关系，对未完成的工作，依然使用原来的预算值。那么，对于最终估算成本就是已完成工作的实际成本加上未完成工作的预算成本。

$$EAC=ACWP+(BAC-BCWP)$$

（3）第三种方法是重新对未完成的工作进行预算工作，这需要一定的工作量。当使用这种方法时，实际上是对计划中的成本预算的否定，认为需要进行重新预算。

$$EAC=ACWP+\text{重新进行的成本预算}$$

3）成本控制的措施与方法

控制软件项目成本的措施归纳起来有三大方面：组织措施、技术措施和经济措施。三者是融为一体、相互作用的。

项目经理是项目成本控制的中心，要以投标报价为依据，制定项目成本控制目标，通过各部门和项目组各成员的努力合作，形成以市场投标报价为基础，实施方案经济优化、设备采购经济优化、人员配备经济优化的项目成本控制体系。

软件成本控制措施，如图 10-26 所示。动态成本控制原理，如图 10-27 所示。动态成本控制流程，如图 10-28 所示。

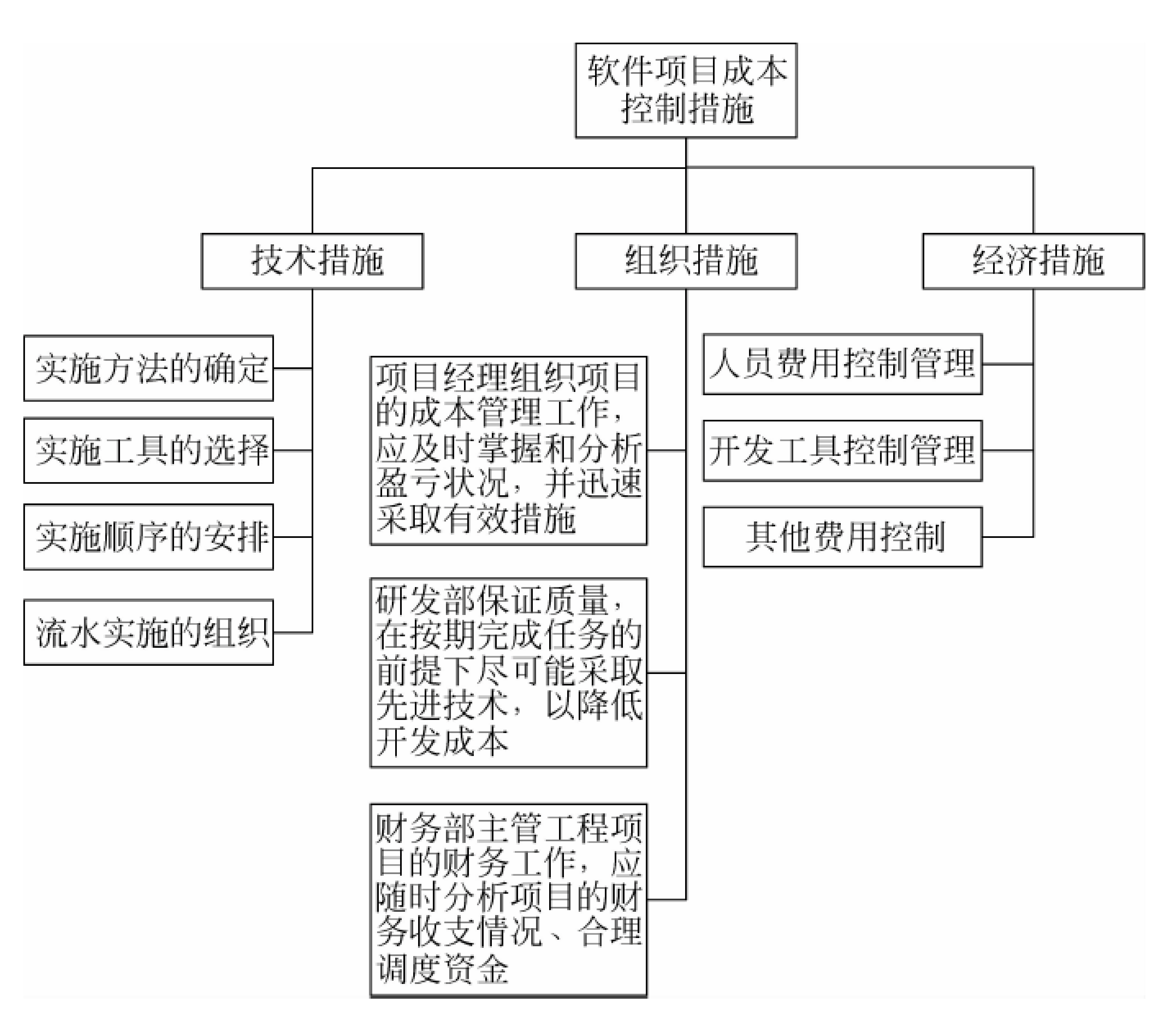


图 10-26　软件成本控制措施图

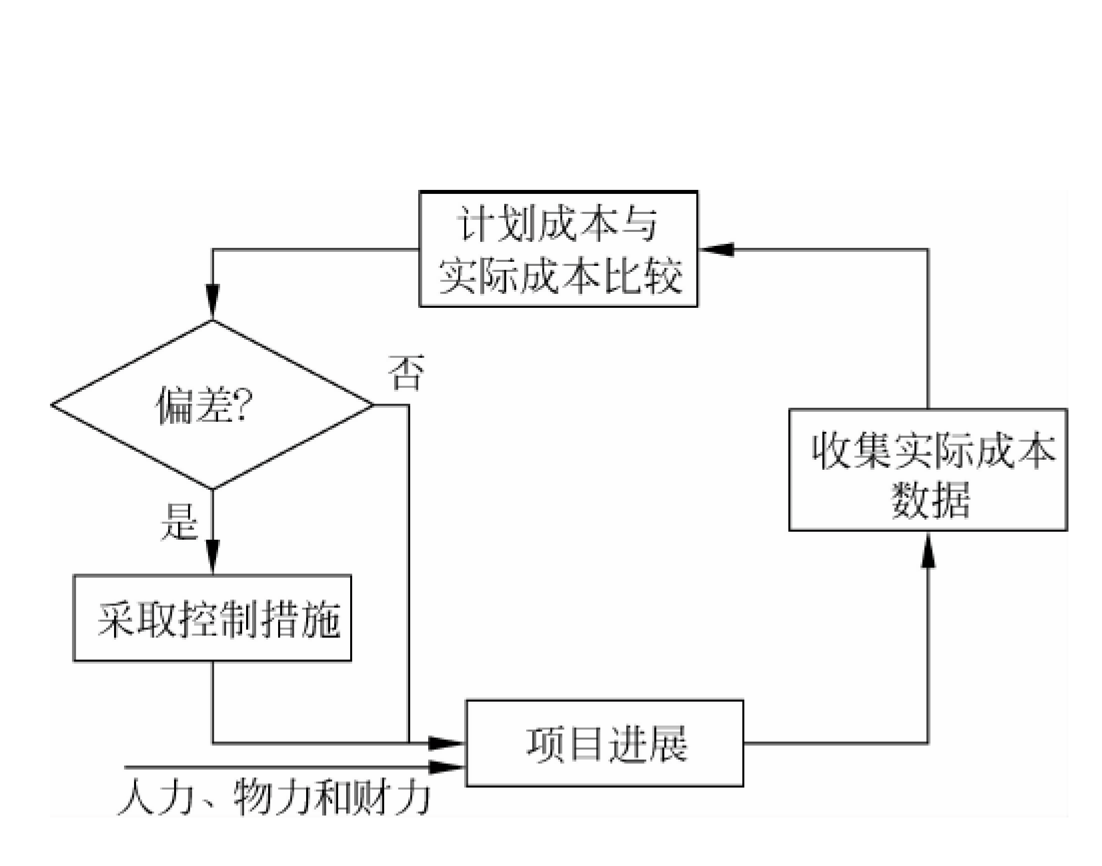


图 10-27　动态成本控制原理示意图

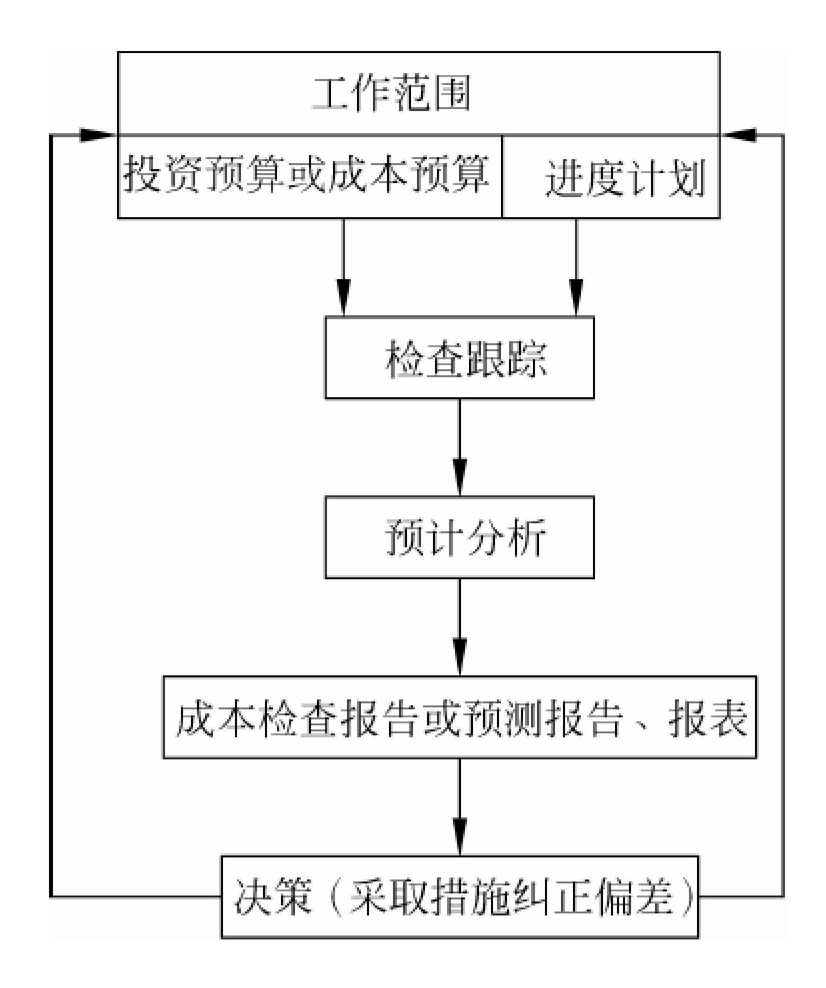


图 10-28　动态成本控制流程

10.4.3　软件项目时间管理

软件项目时间管理指在项目的进程中，为了确保项目能够在规定的时间内实现目标，对项目活动进度及日程安排所进行的管理过程，如图 10-29 所示。时间管理常被认为是产生项目冲突的主要根源，因为大多数项目都超过了之前的时间估计。

时间管理涉及的主要过程包括活动定义、活动排序、活动历时估算、进度计划制定和进

度控制。

- 活动定义：确定项目团队成员和项目干系人为完成项目可交付成果而必须完成的具体活动。
- 活动排序：确定项目活动之间的关系或依赖关系，并形成相应的文档。
- 活动历时估算：对完成各项活动所花费的时间进行估算。这些时间估算包括实际工作时间加间歇时间。
- 进度计划制定：分析活动顺序、活动历时估算和资源要求，制定项目进度计划。
- 进度控制：控制和管理项目计划的变更。

时间管理一般包括相互影响的三个环节。

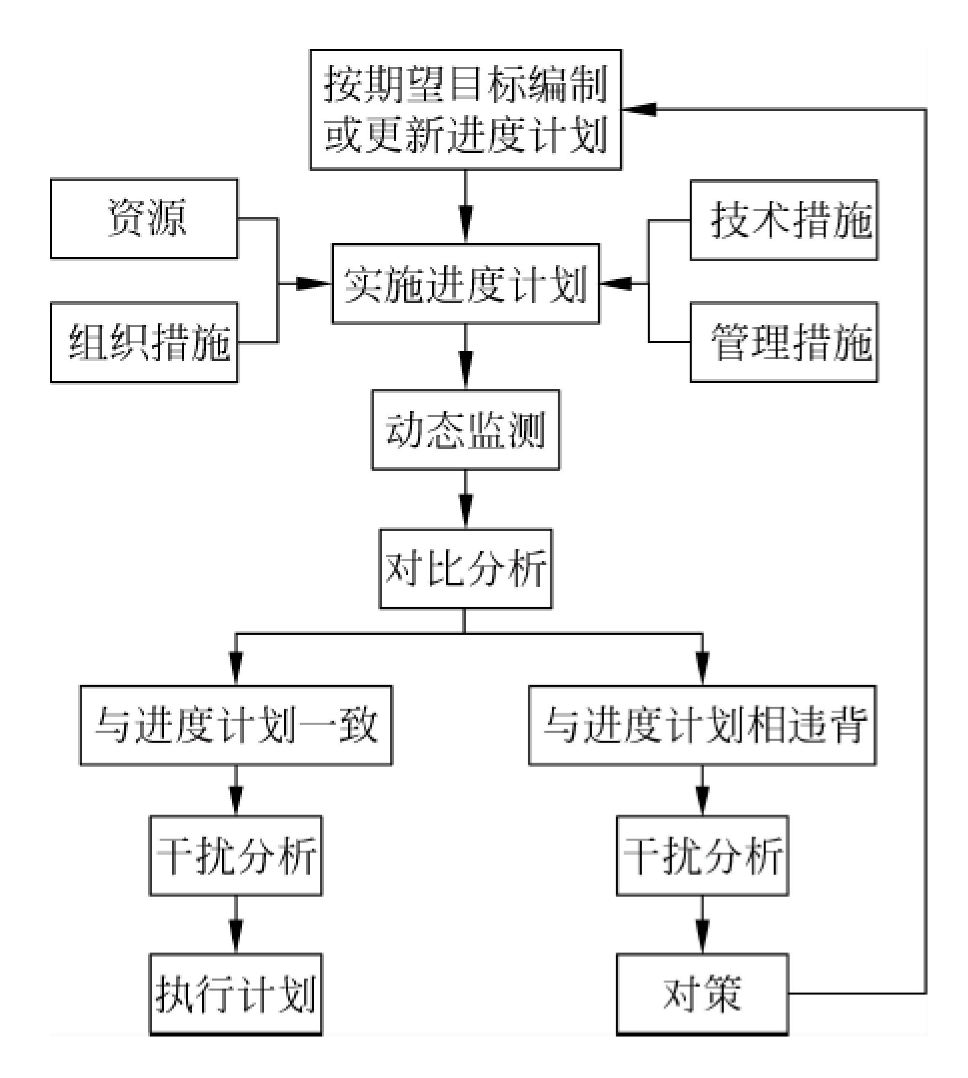


图 10-29　软件项目时间管理

- 进度计划是时间管理的基础，时间管理是通过项目的动态监控实现的。
- 项目时间管理是随着项目的进行而不断进行的，是一个动态过程，也是一个循环进行的过程。
- 对比分析并采取必要的措施是时间管理的关键。

1. 活动定义

活动定义是一个过程，它涉及确认和描述一些特定的活动，完成了这些活动意味着完成了 WBS 结构中的项目细目和子细目，如图 10-30 所示。通过定义活动过程可使项目目标体现出来。软件项目的活动是软件项目为产生各个可交付成果（如代码）所必须进行的具体活动，其目的是将软件项目工作分解为更小、更易管理的工作包，也叫活动或任务。这些小的活动应该是能够保障完成交付软件产品的可实施的详细任务。

软件项目活动定义	需求分析
	系统分析
	设计
	开发编码
	测试
	实施

图 10-30　活动定义

项目活动定义所需要的信息包括 WBS、项目范围的界定、历史信息、项目前期收集和积累的各种信息、项目组织或他人过去开展的类似项目信息、项目的约束条件、项目的假设前提。

项目活动定义的结果是给出下述信息文件：

- 项目活动清单。项目活动清单必须列一个项目所需要开展的全部活动。
- 相关的支持细节。支持和说明项目活动清单的各种具体细节文件与信息。
- 更新的 WBS。当出现这种情况的时候，还需要同时更新相关的项目管理文件，如项目的成本估算文件等。

工作分解结构（WBS）将软件项目逐层分解成一个个可执行的任务单元，这些任务单元既构成了整个项目的工作范围，又是进度计划、人员分配和成本计划的基础，如图 10-31 所示。软件项目的 WBS 以可交付软件产品为导向对软件项目的过程要素进行分组，它归纳和定义了项目的整个工作范围。

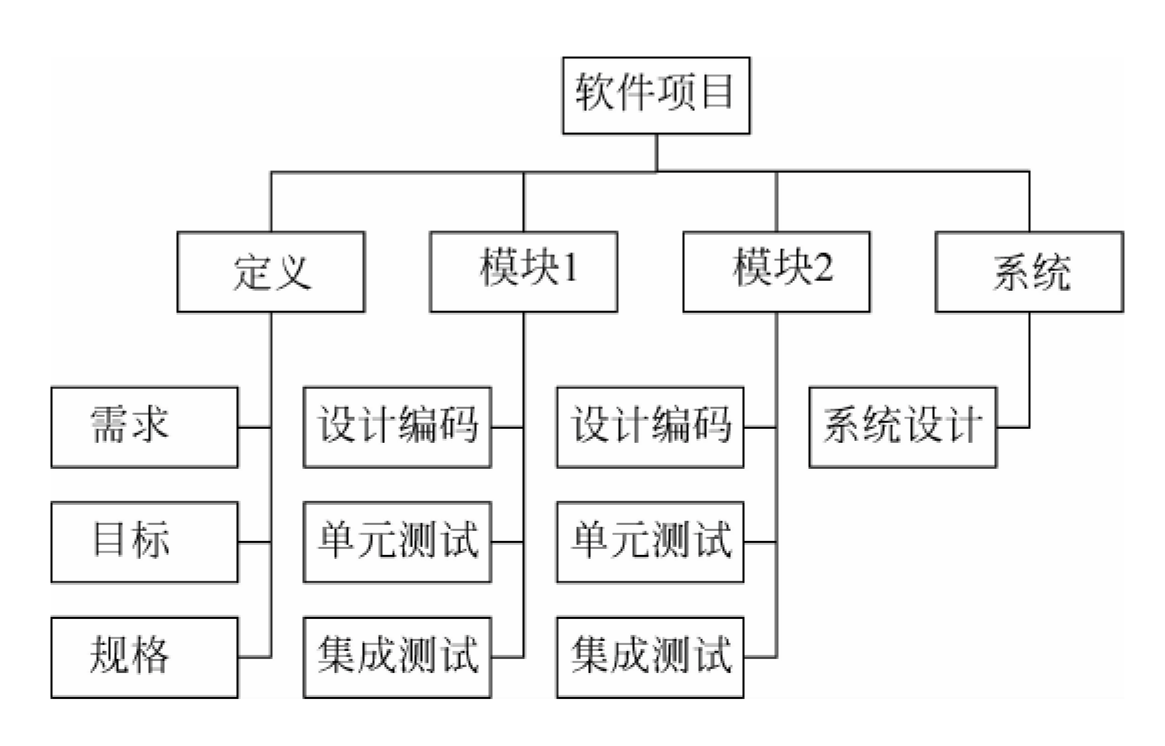


图 10-31 典型软件项目 WBS

WBS 可以由树形的层次结构图或者行首缩进的表格表示，每下降一层代表对项目工作的更详细的定义，如图 10-32 所示。

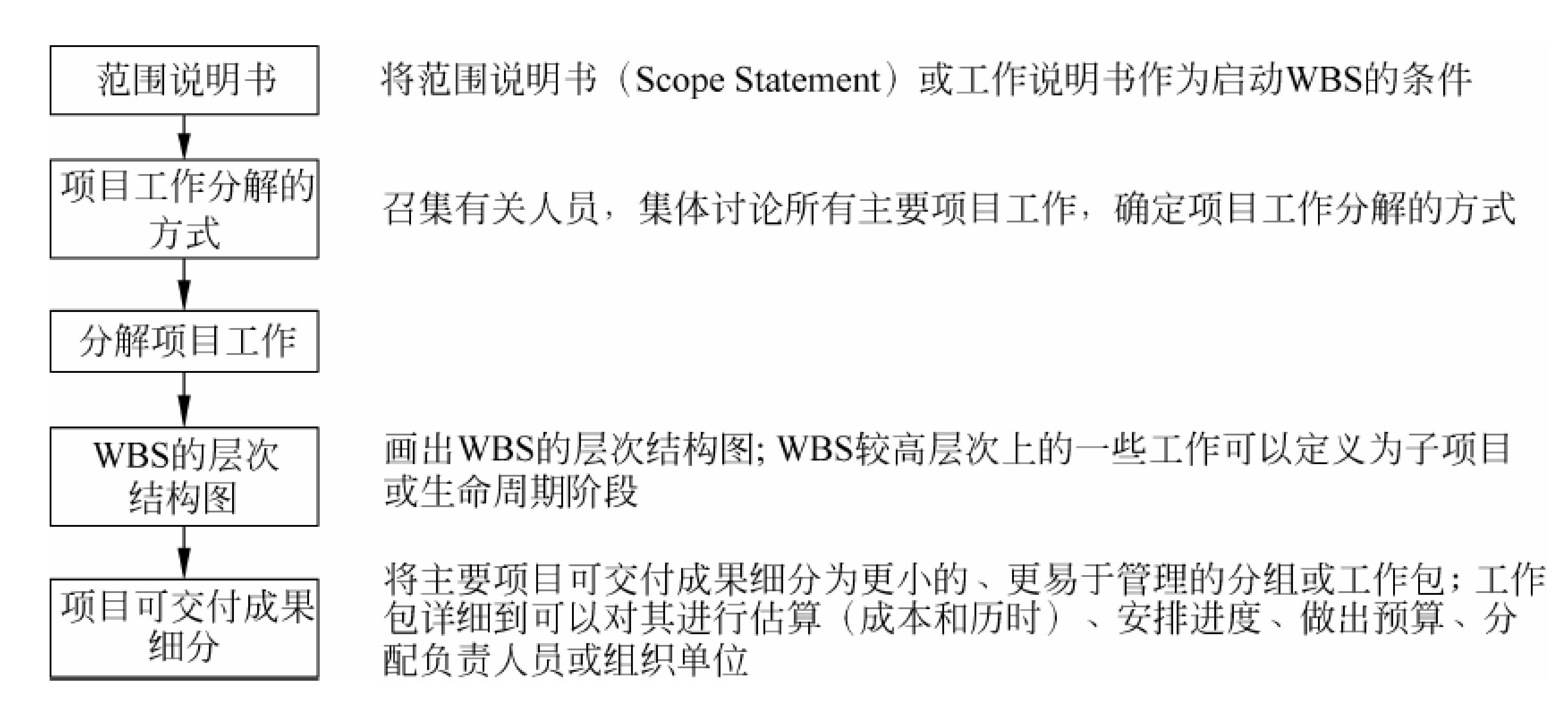


图 10-32 WBS 的编制过程

2. 活动排序

项目活动排序是指识别项目活动清单中各项活动的相互关联与依赖关系，并据此对项目各项活动进行先后顺序的安排和确定工作。

项目活动排序所需要的信息有：

- 项目活动清单及其支持细节
- 项目产出物描述
- 项目活动之间的必然依存关系
- 项目活动之间的人为依存关系
- 项目活动的外部依存关系
- 项目的约束与假设条件

活动之间的四种依赖关系(见图 10-33)为：

- 完成-开始(Finish-Start)。A 活动必须在 B 活动开始前完成。
- 开始-开始(Start-Start)。A 活动必须在 B 活动开始前开始。

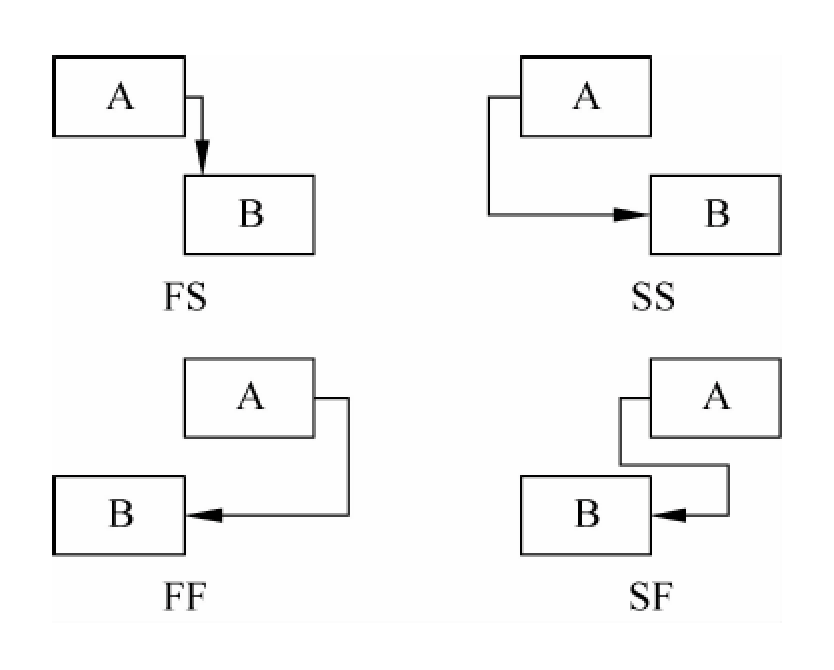
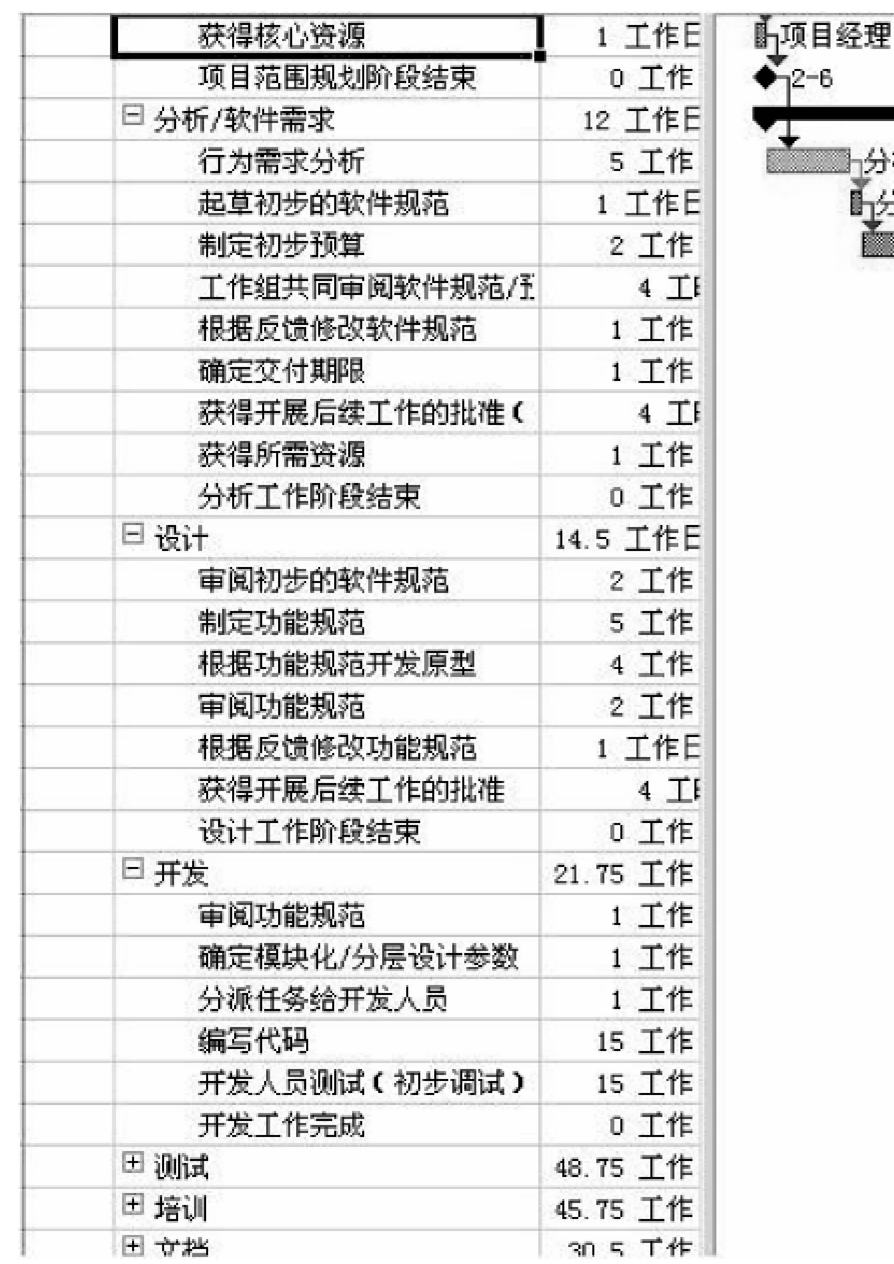


图 10-33 项目活动排序

- 完成-完成(Finish-Finish)。A 活动必须在 B 活动完成前完成。
- 开始-完成(Start-Finish)。A 活动必须在 B 活动完成前开始。

1) Gantt 图(甘特图)

甘特图也被称为线条图,它是以横线来表示每项活动的起止时间的,如图 10-34 所示。甘特图简单、明了、直观、易于编制,是在小型项目中人们常用的工具。即使在大型工程项目中,它也是高级管理层了解全局、基层安排进度时有用的工具。



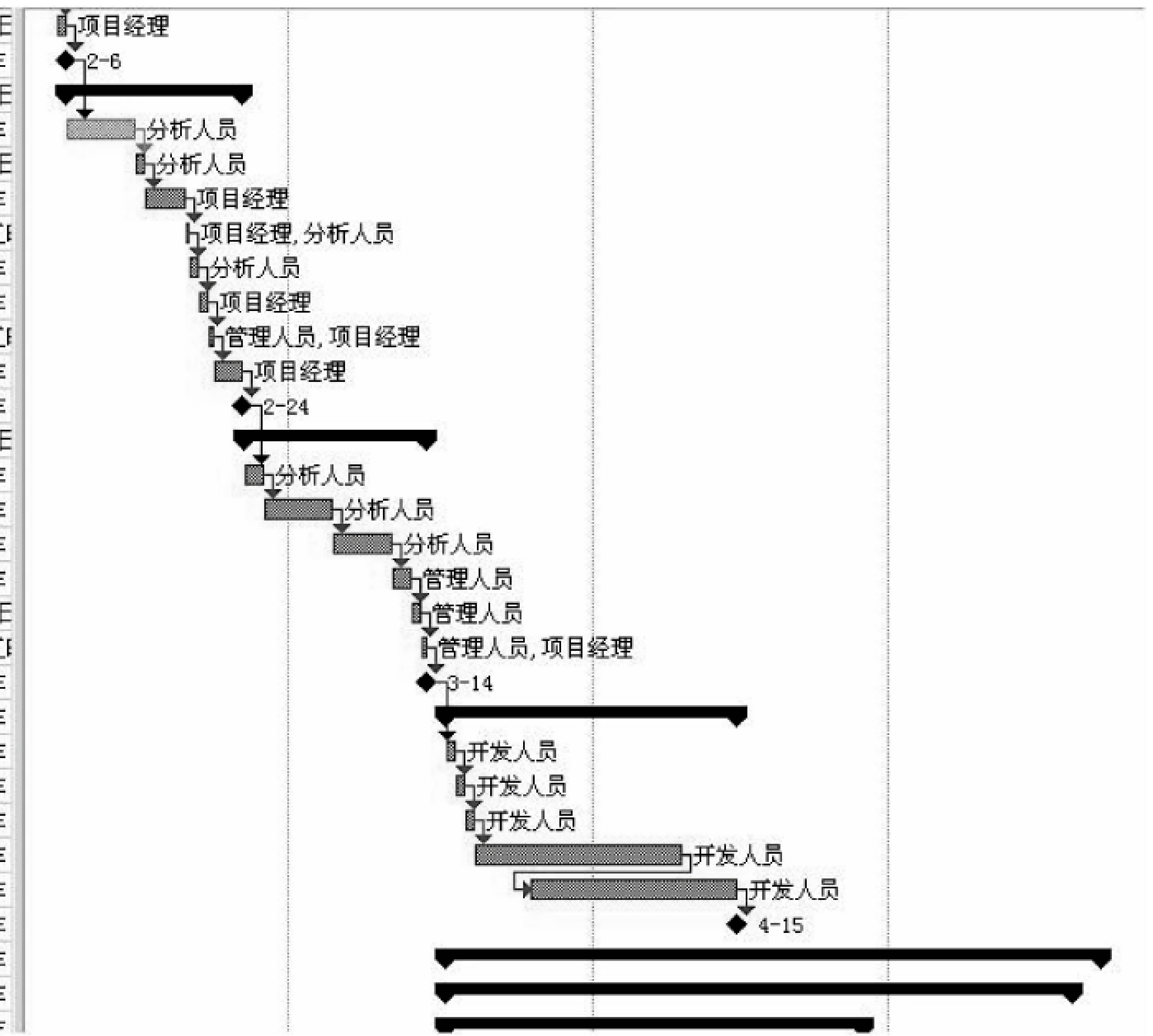


图 10-34　甘特图

甘特图能很形象地描绘任务分解情况,以及每个子任务(作业)的开始时间和结束时间,因此是进度计划和进度管理的有力工具。它具有直观、简明、容易掌握、容易绘制的优点,但是甘特图也有三个主要缺点。

- 不能显示地描绘各项作业彼此间的依赖关系。
- 进度计划的关键部分不明确,难以判定哪些部分应当是主攻和主控的对象。
- 计划中有潜力的部分及潜力的大小不明确,往往造成潜力的浪费。

当把一个软件工程项目分解成许多子任务,并且它们彼此间的依赖关系又比较复杂时,仅仅用甘特图作为安排进度的工具是不够的,不仅难以做出既节省资源又保证进度的计划,而且还容易发生差错。

2) 网络图

网络图是制定进度计划时另一种常用的图形工具,同样能描绘任务分解情况以及每项作业的开始时间和结束时间。网络图将任务计划和进度安排分开的职能是甘特图所没有的,因此,一旦某项活动的时间延误,甘特图整体将面临很大变动,而网络图则不然。此外,它还能显示地描绘各个作业彼此间的依赖关系。

网络图分为前导图法和箭线图法。网络图中的工作是指按需要的粗细程度将计划任务划分而成的、消耗时间或同时也消耗资源的一个子项目或子任务。工作可以是软件项目，也可以是软件项目中的模块工作。

- 前导图法(Precedence Diagramming Method，PDM)。也被称为单代号网络图法(Activity On Node，AON)，是指按工作先后顺序把每项工作作为一个结点，用结点之间的箭线表示项目活动之间的相互关系，如图 10-35 所示。

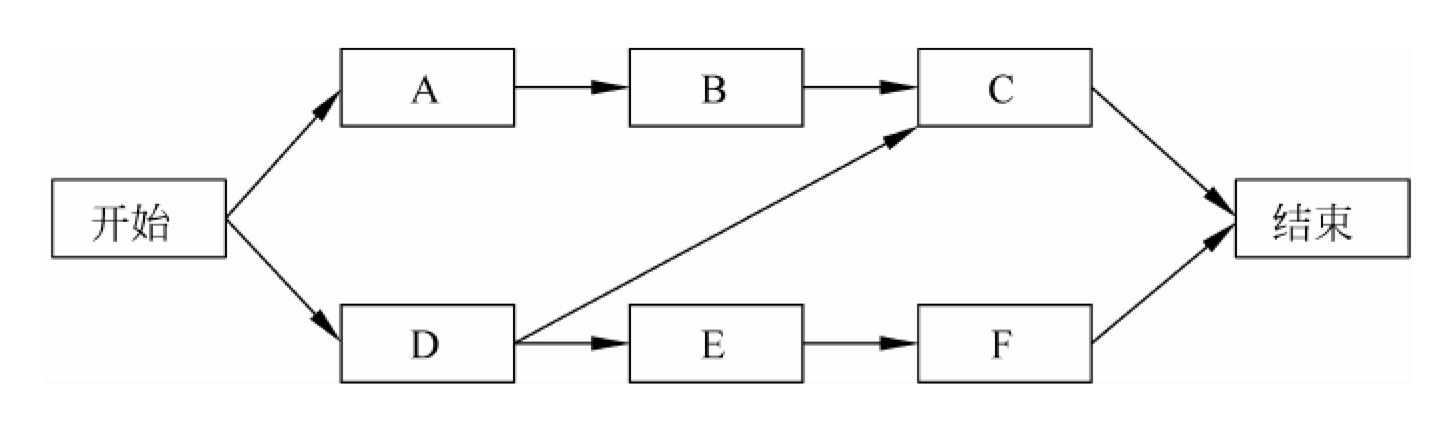


图 10-35 单代号网络图

- 箭线图法(Arrow Diagramming Method，ADM)。箭线图法也被称为双代号网络图法(Activity On Arc，AOA)，是一种利用箭线代表活动，而用结点代表活动之间的联系和相互依赖关系的编制项目网络图的方法。双代号网络图与单代号网络图的区别是后者把工作放在结点上，如图 10-36 所示。

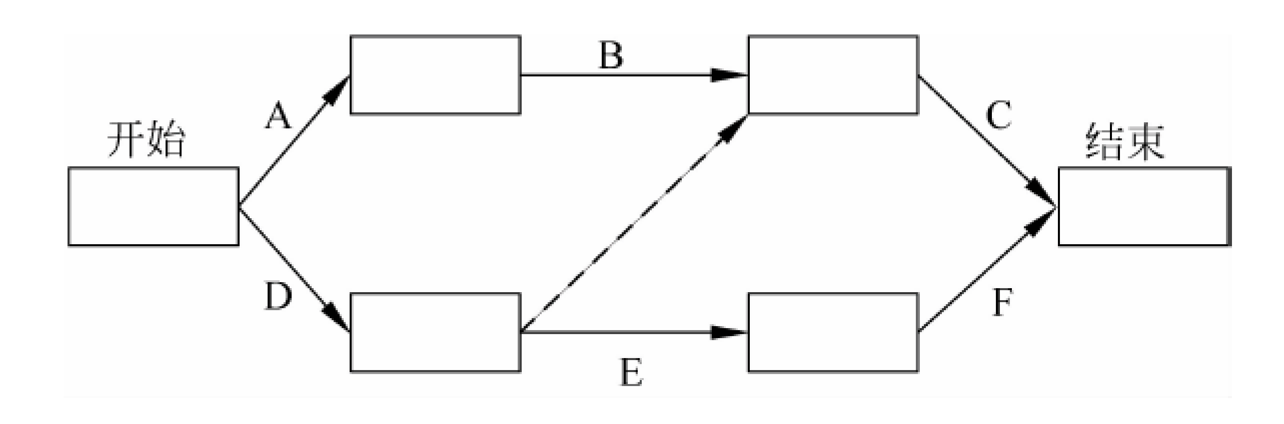


图 10-36 双代号网络图

箭线图可描述的活动类型如图 10-37 所示。

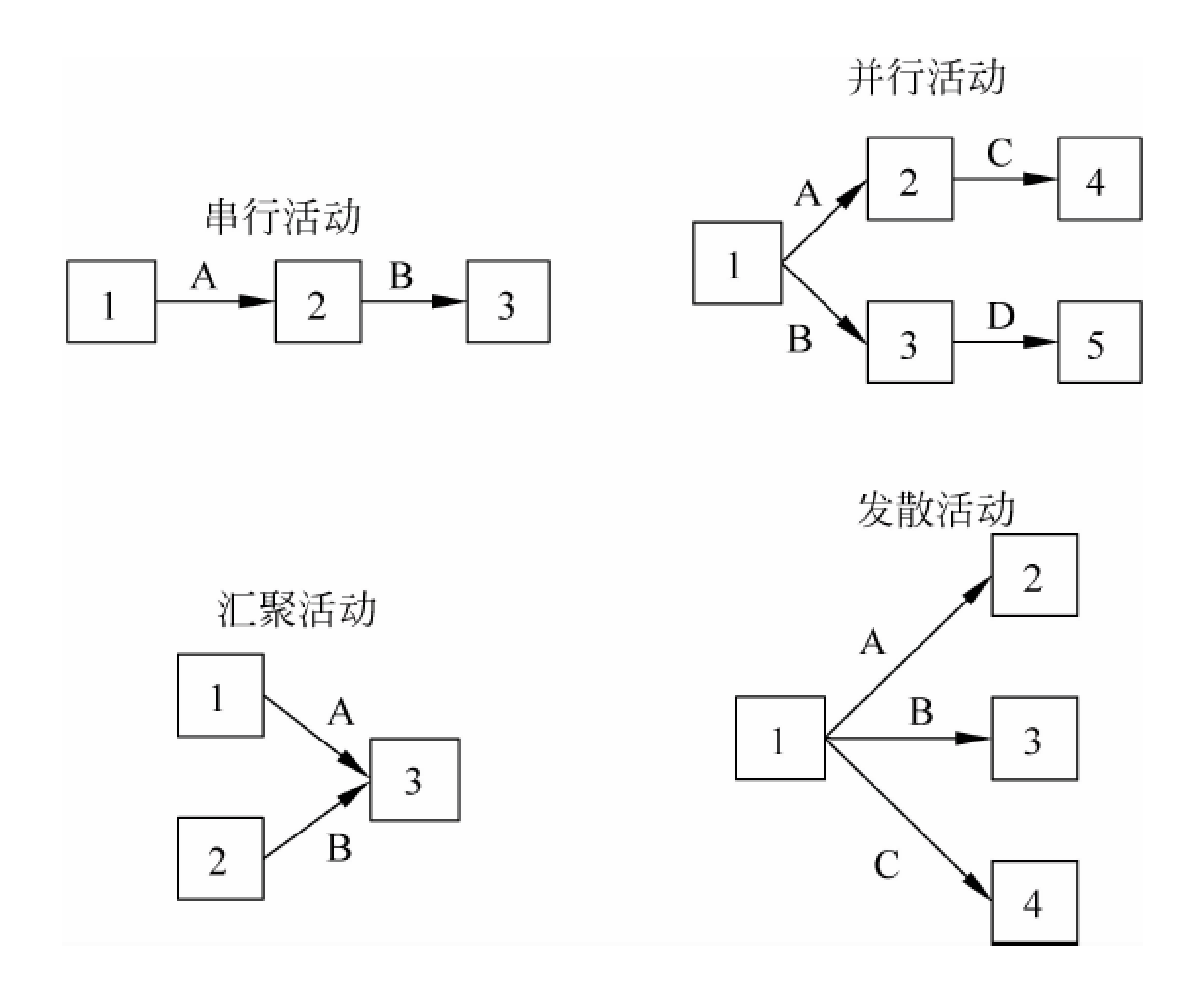


图 10-37 活动类型

箭线式网络图的绘制规则有：是有向图，不能出现回路；两个相邻结点间只允许有一条箭线直接相连；箭线只能从一个结点开始，到另一结点结束；每个网络必须也只能有一个起点事项和一个终点事项；避免使用反向箭线；不允许出现双箭头或无箭头的线。

项目活动排序的工作结果为项目网络图和更新后的项目活动清单。

3. 活动历时估算

活动历时估算是指对项目已确定的各种活动所做出的可能工期长度的估算工作。活动工期估算的依据包括：

- 项目活动清单；
- 项目的约束和假设条件；
- 项目资源的数量要求；
- 项目资源的质量要求；
- 历史信息。

计划评审技术(Program Evaluation and Review Technique，PERT)是 20 世纪 50 年代末美国海军部门开发北极星潜艇系统时为协调三千多个承包商和研究机构而开发的，其理论基础是假设项目持续时间以及整个项目完成时间是随机的，且服从某种概率分布。PERT 可以估计整个项目在某个时间内完成的概率。

如果对一项工作估计缺乏足够的信息，或者说考虑到未来环境的变化，对它的时间估计不能一次进行，这时可以采用三点估计法。PERT 对各个项目活动的完成时间按三种不同情况估计(见图 10-38)：

- 乐观时间(Optimistic Time)：任何事情都顺利的情况下完成某项工作的时间。
- 最可能时间(Most Likely Time)：正常情况下完成某项工作的时间。
- 悲观时间(Pessimistic Time)：最不利的情况下完成某项工作的时间。

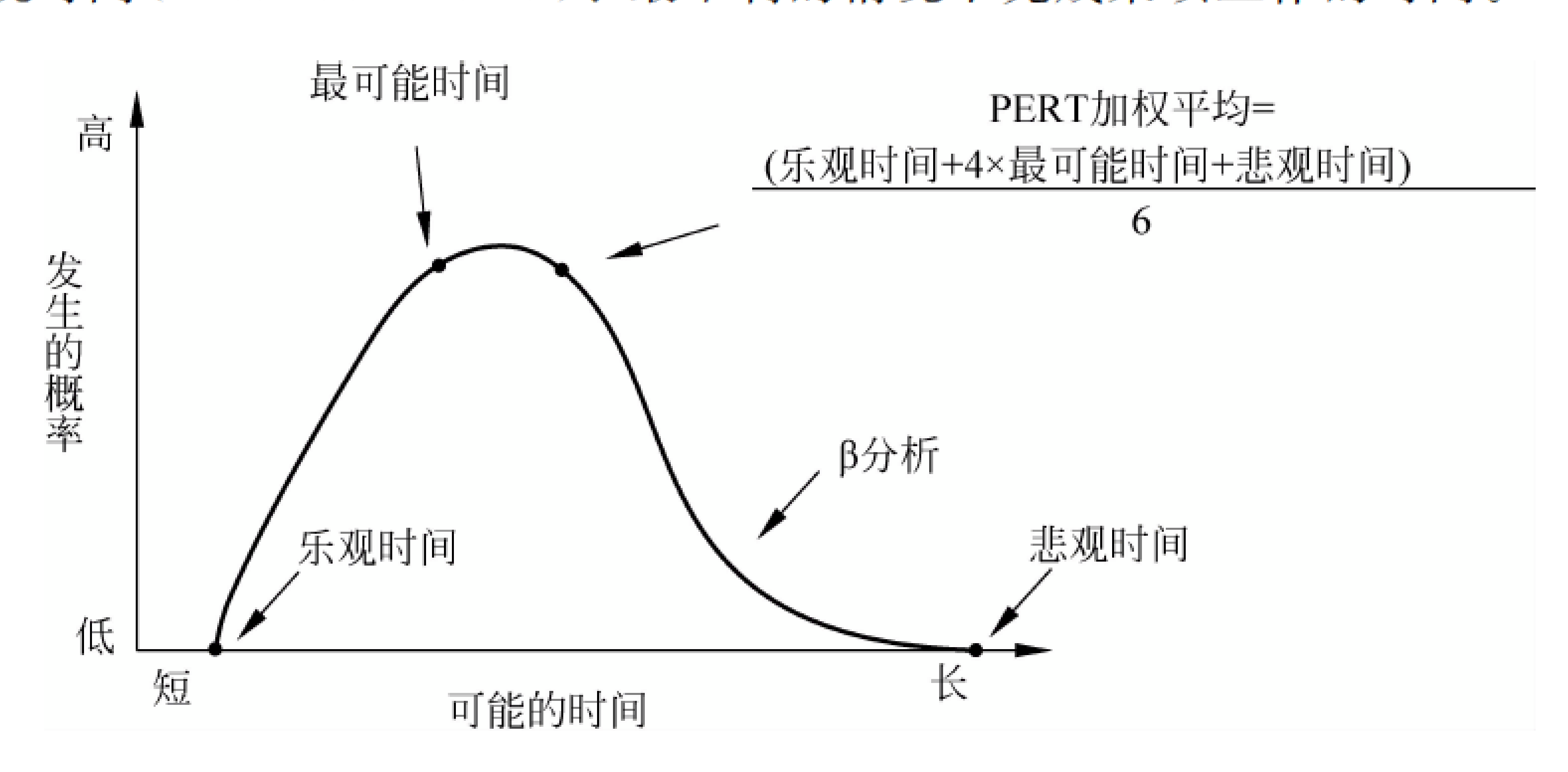


图 10-38　单项活动的 PERT 所需时间估算

假定三个估计服从 β 分布，由此可算出每个活动的期望 t_i：

$$t_i = \frac{a_i + 4m_i + b_i}{6}$$

其中 a_i表示第 i 项活动的乐观时间，m_i表示第 i 项活动的最可能时间，b_i表示第 i 项活动的悲观时间。

根据 β 分布的方差计算方法，第 i 项活动的持续时间方差为：

$$\sigma_i^2 = \frac{(b_i - a_i)^2}{36}$$

PERT 认为整个项目的完成时间是各个活动完成时间之和，且服从正态分布。

活动历时估算阶段将估算出项目活动工期，并提交项目工期估算的依据和更新后的活动清单。

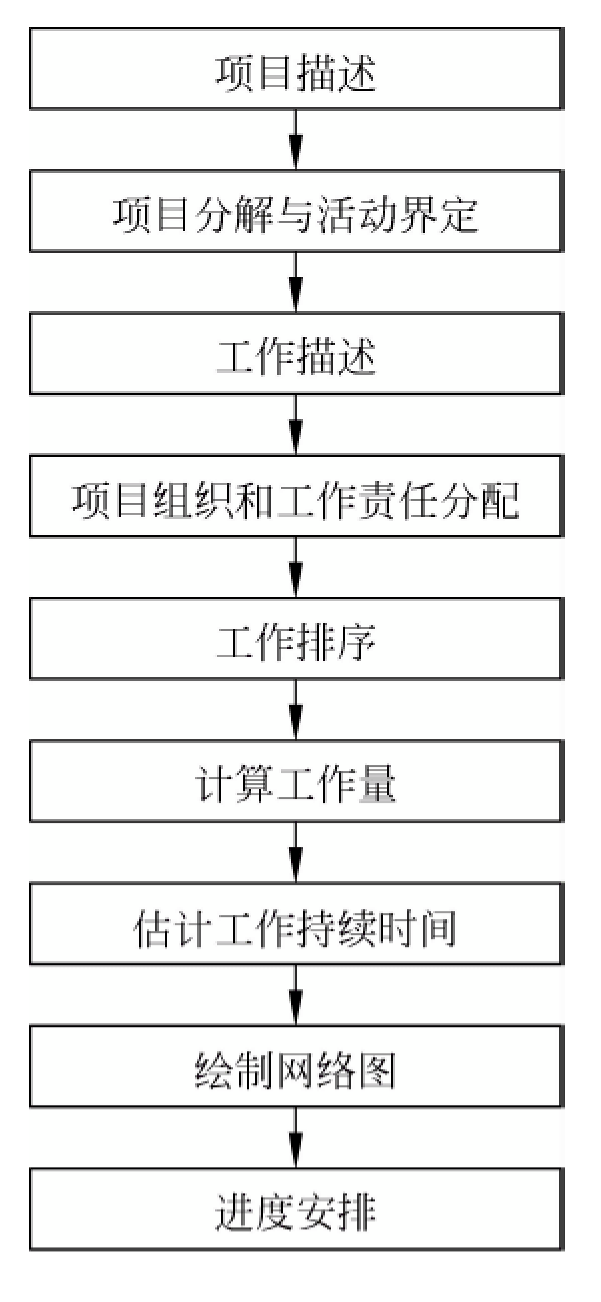


图 10-39 进度计划制定的步骤

4. 进度计划制定

在软件项目中，进度计划的制定包括了项目描述、项目分解与活动界定、工作描述、项目组织和工作责任分配、工作排序、计算工作量、估计工作持续时间、绘制网络图、进度安排等活动，如图 10-39 所示。

关键路径就是在项目网络图中，决定项目最早完成时间的所有活动形成的路径，也是项目网络图中最长的路径。关键路径可能随着某些活动提前完成或延迟完成而改变。关键路径上的活动具有最少的浮动时间，即总时差为 0。处于关键路径上的活动被称为关键活动(Critical Activity)。

关键路径法(Critical Path Method，CPM)是用来确定哪些活动组成的路径具有最少总时差的一种网络分析技术，通过它可以确定项目何时能完成，如图 10-40 所示。这种方法借助网络图和各活动所需时间(估计值)，计算每一活动的最早或最迟开始和结束时间。CPM 法的关键是计算总时差，这样可决定哪一活动具有最小时间弹性；核心思想是将 WBS 分解的活动按逻辑关系加以整合，统筹计算出整个项目的工期和关键路径。

每个活动有四个和时间相关的参数。

- 最早开始(Earliest Start，ES)时间：根据网络逻辑和进度限制，未完成部分的一项活动最早可能开始的时间。最早开始时间会随着项目进展和项目计划的变化而变化。
- 最早完成(Earliest Finish，EF)时间：完成一项活动最早的可能时间。

EF＝ES＋工期估计

- 最晚开始(Latest Start，LS)时间：为了使项目按时完成，一项活动必须开始的最迟时间。
- 最晚完成(Latest Finish，LF)时间：为了使项目按时完成，一项活动必须完成的最迟时间。

LS＝LF－工期估计

5. 进度控制

软件项目进度计划控制的目的是增强项目进度的透明度，以便当项目进展与项目计划出现严重偏差时可以采取适当的纠正或预防措施。已经归档和发布的项目计划是项目控制和监督中活动、沟通、采取纠正和预防措施的基础。

1）前提

软件项目进度控制的前提是有效地进行项目计划和充分掌握第一手实际信息，在此前

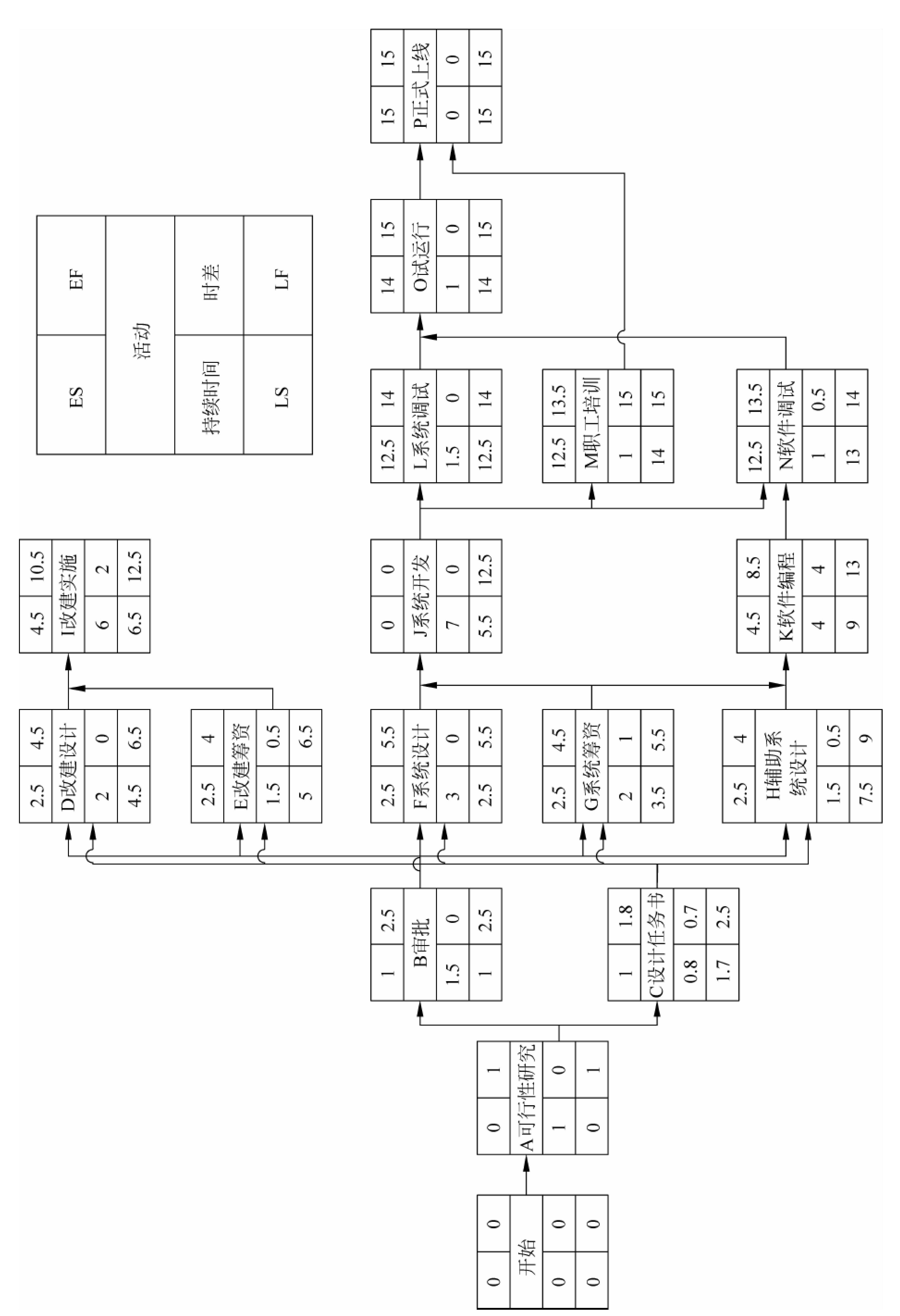


图 10-40 软件项目 CPM 图示例

提下,通过对实际值和计划值进行比较,检查、分析、评价项目进度。通过沟通、肯定、批评、奖励、惩罚、经济等不同手段,对项目进度进行监督、督促、影响、制约;及时发现偏差,及时予以纠正;提前预测偏差,提前予以预防。

2)主要手段

在当前的软件项目开发过程中,无论是开发人员还是管理人员都越来越注意到项目进度的重要性,因此在控制项目进度时,应考虑下面几个方面。

- 加强对供应商项目进度的管理;
- 关注薄弱环节,实现动态平衡;
- 明确每个成员的责任;
- 项目经理与成员的沟通与交流;
- 项目计划书;
- 项目阶段情况汇报与计划;
- 监督机制。

3)内容

从内容上看,软件开发项目进度控制主要表现在组织管理、技术管理、信息管理等方面。组织管理包括以下几个内容。

- 项目经理监督并控制项目进展情况;
- 进行项目分解,如按项目结构分、按项目进展阶段分、按合同结构分,并建立编码体系;
- 制订进度协调制度,确定协调会议时间、参加人员等;
- 对影响进度的干扰因素和潜在风险进行分析。

4)工作要点

进度控制要真正有效,就必须:

- 要有明确的目的;
- 要及时;
- 要考虑代价;
- 要适合项目实施组织和项目班子的特点;
- 要注意预测项目过程的发展趋势;
- 要有灵活性;
- 要有重点;
- 要便于项目干系人了解情况;
- 要有全局观念。

5)措施

项目进度控制措施主要有以下几个。

- 项目计划评审;
- 项目实施保证措施,包括进度计划的贯彻、调度工作、抓关键活动的进度;
- 项目进度动态检测,包括日常观测、定期观测和项目进展报告。

6)不同阶段的项目进度控制

- 准备阶段的进度控制。向客户提供有关项目信息,协助客户确定工期总目标,编制阶段计划和项目总进度计划,控制该计划的执行。

- 需求分析和设计阶段的进度控制。编制与用户的沟通计划、需求分析工作进度计划、设计工作进度计划，控制相关计划的执行等。
- 实施阶段的进度控制。编制实施总进度计划并控制其执行，编制实施计划并控制其执行等，与客户协调进度计划的编制、调整并采取措施确保进度目标的实现。

10.4.4　软件项目人力资源管理

软件项目成功的关键是有高素质的软件开发人员，然而大多数软件的规模都很大，单个软件开发人员无法在给定期限内完成开发工作。因此，必须把多名软件开发人员合理地组织起来，通过有效的人力资源管理使他们分工协作共同完成开发工作。

软件项目人力资源管理就是根据项目的目标、项目活动进展情况和外部环境的变化，采取科学的方法，对项目团队成员的行为、思想和心理进行有效地管理，充分发挥他们的主观能动性，实现项目的最终目标。

如图 10-41 所示，项目人力资源管理的重点集中在两个方面：个人和团队。

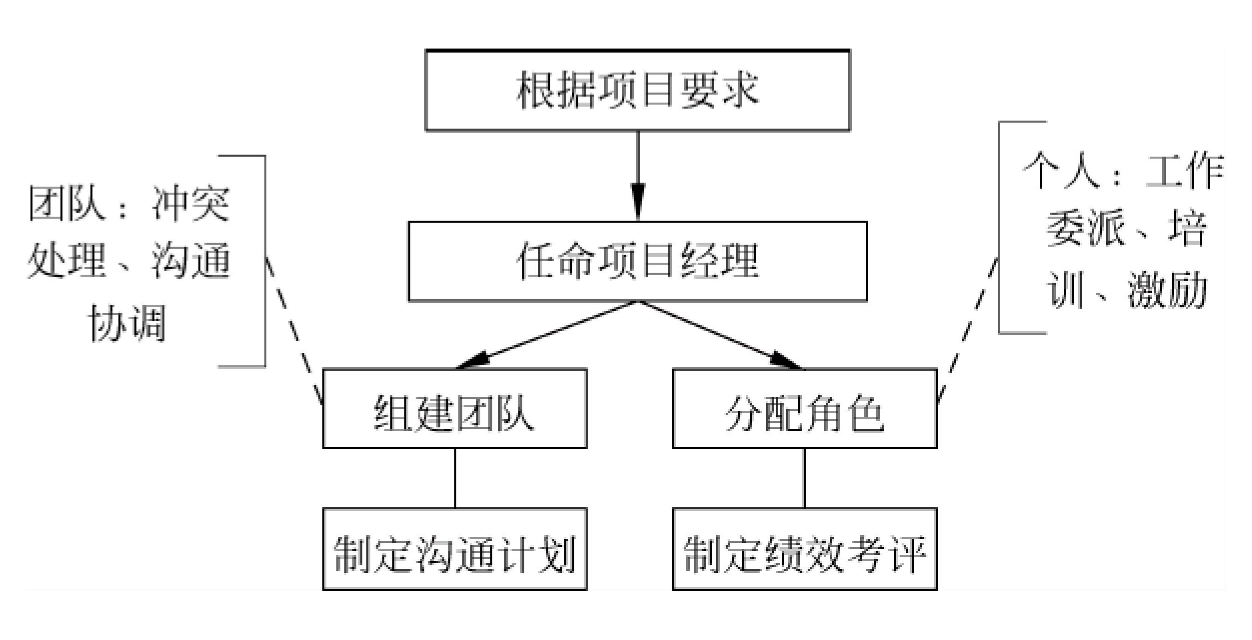


图 10-41　项目人力资源管理

软件开发是智力密集、劳动密集型项目，受人力资源影响最大，项目成员的结构、责任心、能力和稳定性对项目的质量以及是否成功有着决定性的影响。

软件项目人力资源管理的基本内容包括：

- 项目经理的确定；
- 项目组织形式的确定；
- 项目人员的获得与配备；
- 项目团队建设；
- 沟通管理。

项目人力资源管理是将软件开发组织中的系统分析师、软件设计师、程序员、软件测试员、配置管理人员等组织到一起进行项目开发的过程，主要包括项目组织形式的确定、项目团队建设及项目团队管理。

1. 项目组织形式的确定

1）职能型组织结构

职能型组织结构又称为多线性组织结构，是按职能来组织部门分工，即从高层到基层，均把承担相同职能的管理业务及其人员组合在一起，设置相应的管理部门和管理职务，如图 10-42 所示。

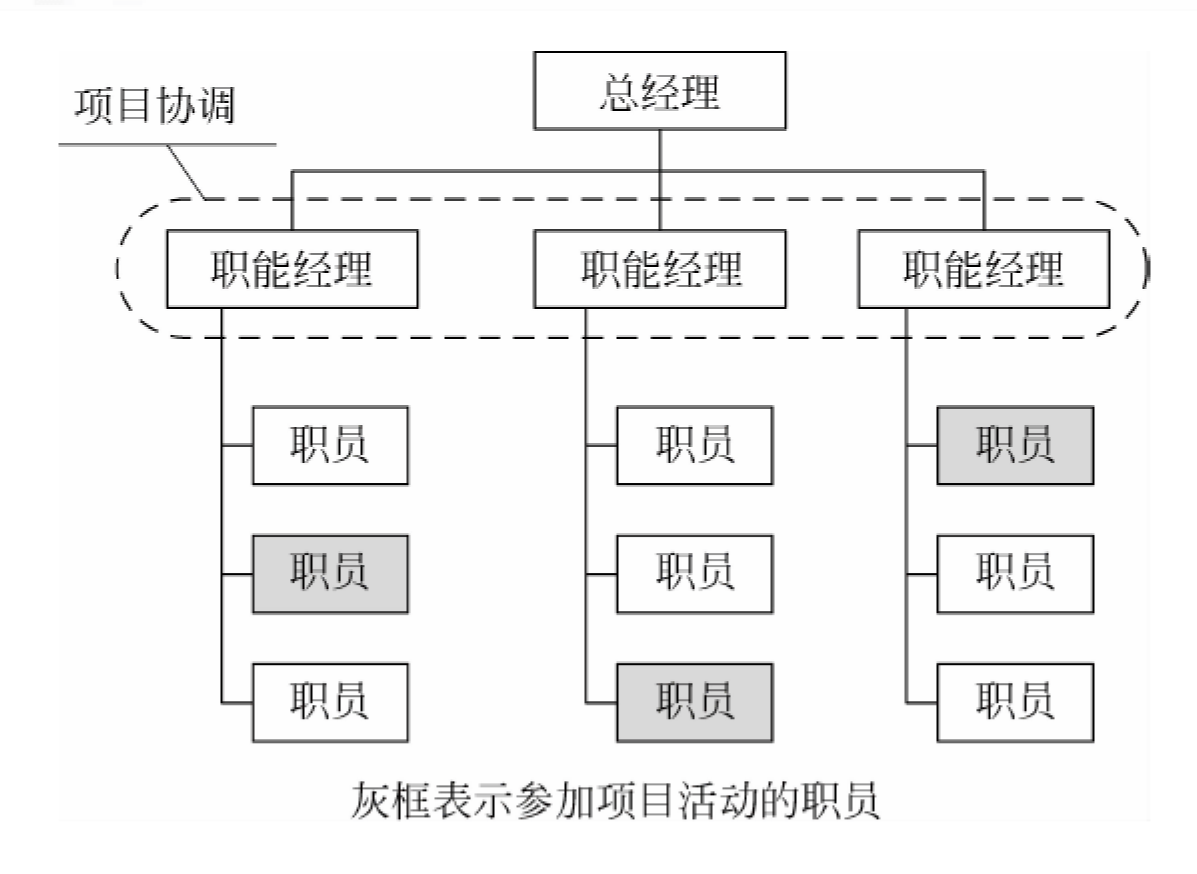


图 10-42 职能型组织结构

职能型组织结构以职能部门作为承担项目任务的主体，可以充分发挥职能部门的资源集中优势，有利于保障项目需要资源的供给和项目可交付成果的质量。职能部门内部的技术专家可以同时被该部门承担的不同项目所使用，节约人力、减少了资源的浪费。同一职能部门内部的专业人员便于相互交流、相互支援，对创造性地解决技术问题很有帮助。当有项目成员调离项目或离开组织，所属职能部门可以增派人员，保持项目的技术连续性。项目成员可以将完成项目和完成本部门的职能工作融为一体，可以减少因项目的临时性给项目成员带来的不确定性。

职能型组织结构的缺点是客户利益和职能部门的利益常常发生冲突，职能部门会为本部门的利益而忽视客户的需求；当项目需要多个职能部门共同完成，或者一个职能部门内部有多个项目需要完成时，资源的平衡就会出现问题；当项目需要由多个部门共同完成时，权力分割不利于各职能部门之间的沟通交流、团结协作；项目成员在行政上仍隶属于各职能部门的领导，项目经理对项目成员没有完全的领导权。

2）项目型组织结构

项目型组织结构中的部门完全是按照项目需要进行设置的，是一种单目标的垂直组织方式。项目经理具有高度独立性、对项目享有完全的领导权，如图 10-43 所示。完成每个项目目标所需的全部资源被完全划分给该项目，完全为该项目服务。

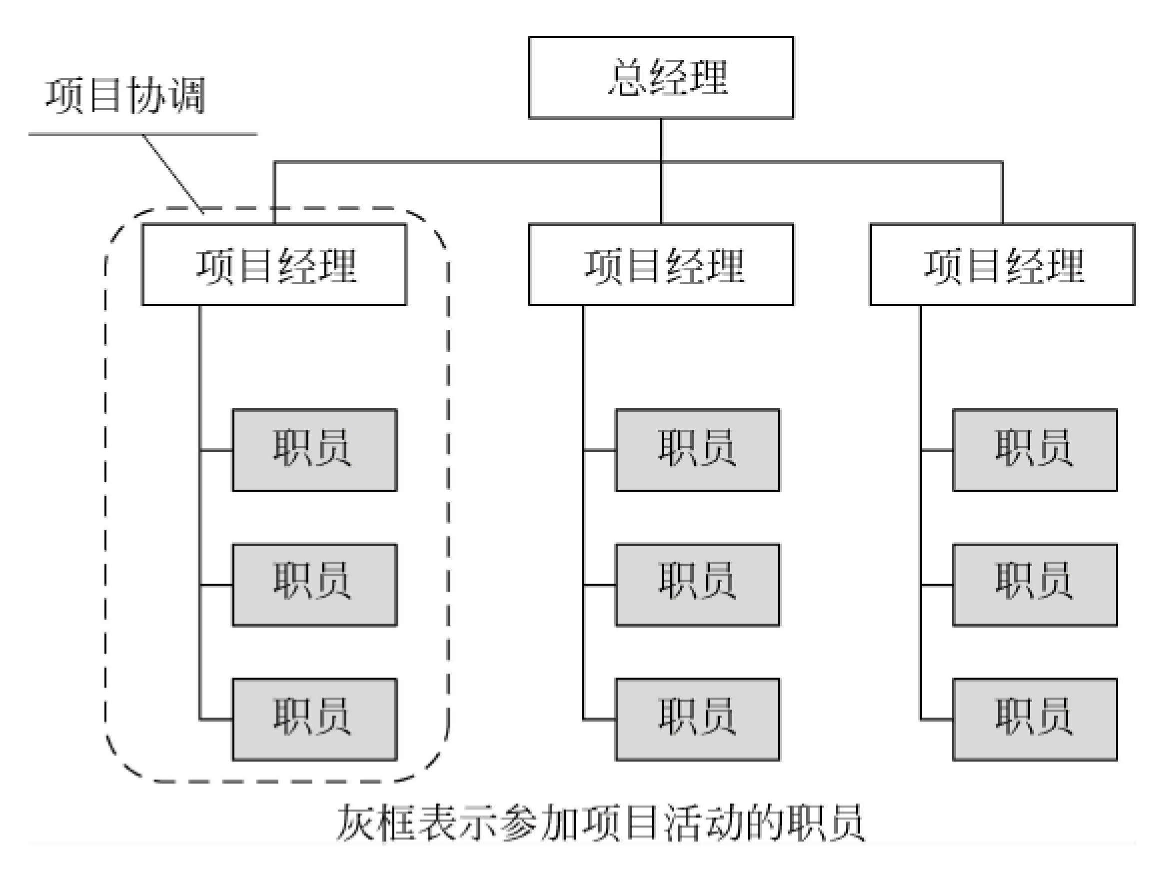


图 10-43 项目型组织结构

在项目型组织结构中，项目经理对项目可以全权负责，可以根据项目需要随意调动项目组织的内部资源或者外部资源。项目型组织的目标单一，完全以项目为中心安排工作，能够对客户的要求做出及时响应，有利于项目的顺利完成。项目经理对项目成员有完全的领导权，项目成员只对项目经理负责，避免了职能型项目组织结构下项目成员处于多重领导、无所适从的局面，项目经理是项目的真正的、唯一的领导者。项目型组织结构简单，项目成员直接属于同一个部门，彼此之间的沟通交流简洁、快速，提高了沟通效率，同时也加快了决策速度。

项目型组织结构的缺点是不同的项目组织，资源不能共享，即使某个项目的专用资源闲置，通常也无法应用于另一个同时进行的项目，人员、设施、设备可能会重复配置，造成一定程度的资源浪费；公司里各个独立的项目型组织处于相对封闭的环境之中，公司的宏观政策、方针很难做到完全地、真正地贯彻实施，可能会影响公司的长远发展；在项目完成以后，项目型组织的使命就完成了，项目成员有可能被解雇，对项目成员来说，缺乏一种事业上的连续性和安全感；软件开发组织承担的项目之间处于一种条块分割状态，项目之间缺乏信息交流。

3）矩阵型组织结构

为解决职能型组织结构与项目型组织结构的不足，发挥它们的长处，人们设计出了介于两种组织结构之间的一种项目组织结构形式，即矩阵型组织结构。在矩阵型项目组织结构中，根据项目的需要，从不同的部门中选择合适的人员组成一个临时项目组，项目结束后，项目组也会立即解体，然后各个成员再回到各自原来的部门。这些人员在项目工作期间，在工作内容上服从项目团队的安排，人员不独立于职能部门之外，是一种暂时的、半松散的组织结构形式，项目团队成员之间的沟通不需要通过其职能部门领导，项目经理往往直接向高层领导汇报工作。根据项目经理对项目的约束程度，矩阵型组织结构又可分成弱矩阵型组织结构、强矩阵型组织结构和平衡矩阵型组织结构三种形式。

（1）弱矩阵型组织结构。一般是指在项目团队中没有一个明确的项目经理，只有一个协调员负责协调工作，如图 10-44 所示。团队各成员之间按照各自职能部门所对应的任务，相互协调进行工作。实际上在这种模式下，相当多的项目经理的职能由职能部门负责人分担了。

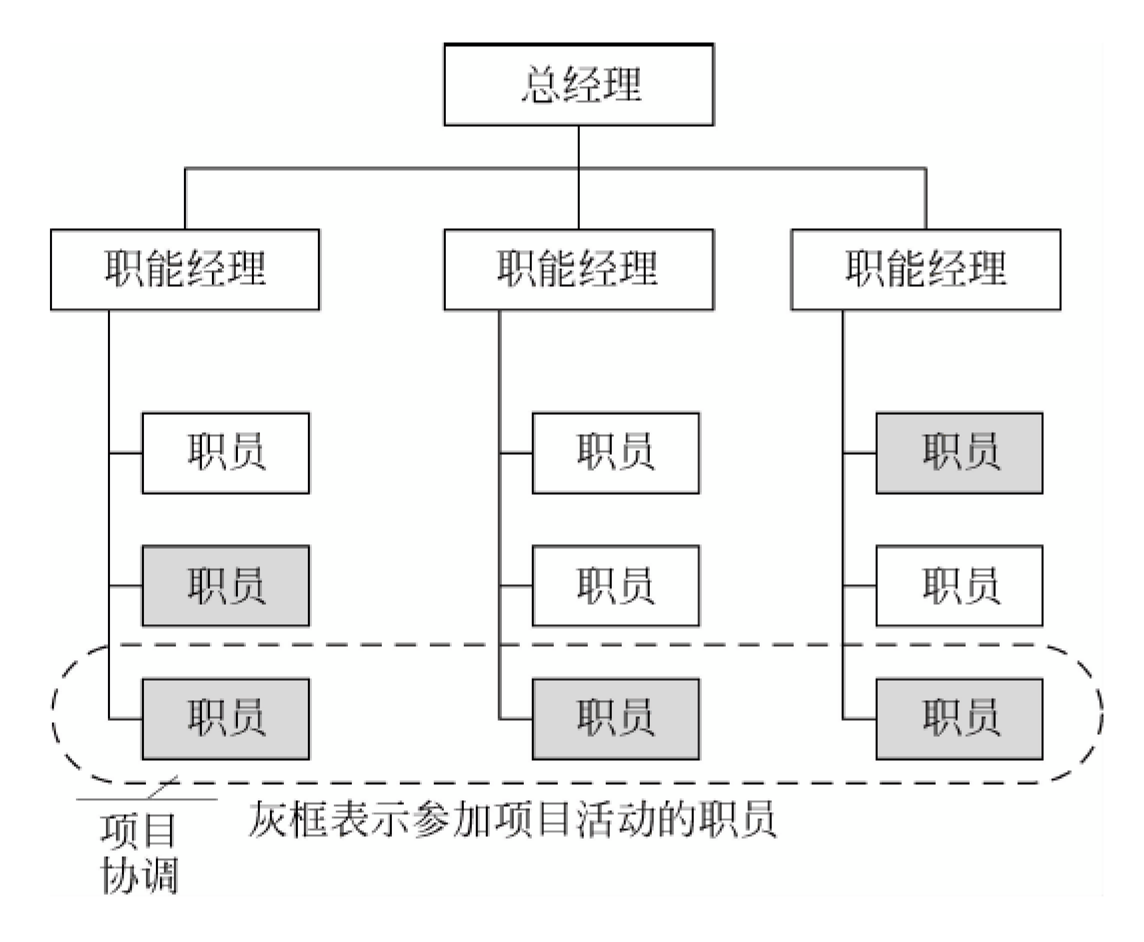


图 10-44 弱矩阵型组织结构

（2）强矩阵型组织结构。这种模式下的主要特点是，有一个专职的项目经理负责项目的管理与运行工作，项目经理通常来自于专门的项目管理部门，如图 10-45 所示。项目经理与上级沟通往往通过其所在的项目管理部门负责人进行。

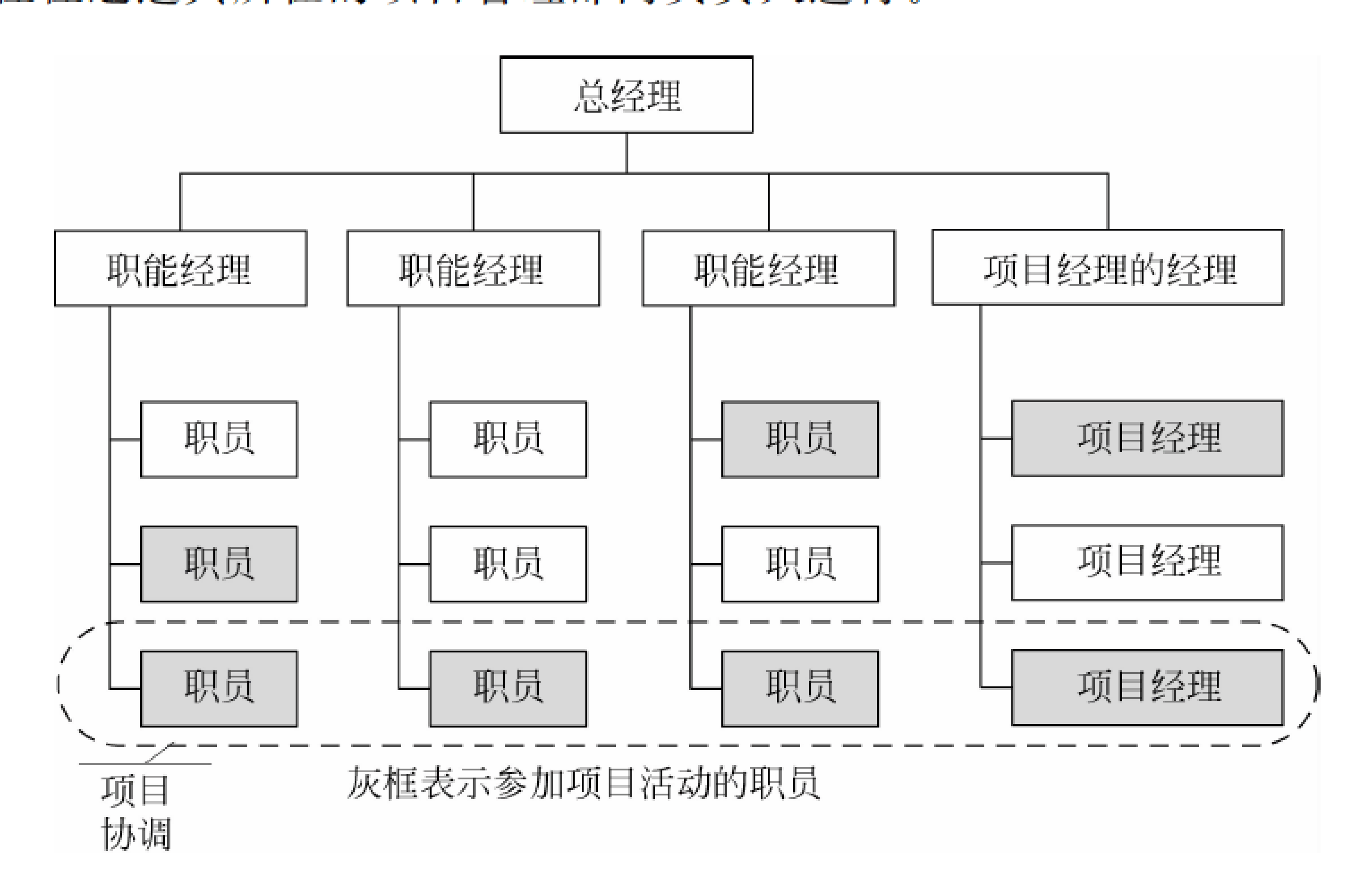


图 10-45　强矩阵型组织结构

（3）平衡矩阵型组织结构。这种组织结构形式是介于强矩阵式与弱矩阵式之间的一种形式，如图 10-46 所示。主要特点是项目经理是由职能部门中的团队成员担任，其工作除项目的管理工作外，还可能负责本部门承担的相应项目任务。此时的项目经理与上级沟通必须在其职能部门的负责人与公司领导之间做出平衡与调整。

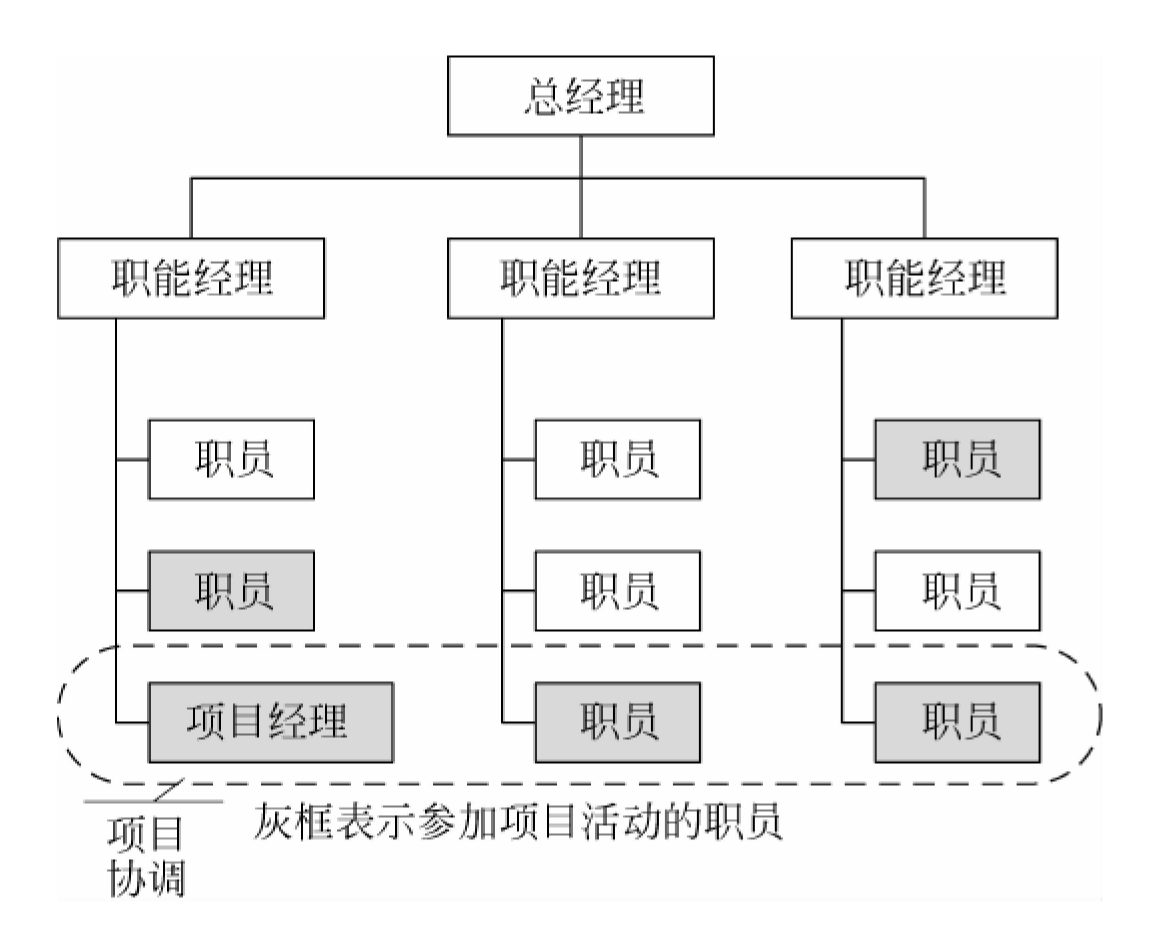


图 10-46　平衡矩阵型组织结构

矩阵型组织结构由专职的项目经理负责整个项目，以项目为中心，因此公司的多个项目可以共享各个职能部门的资源。这样既有利于项目目标的实现，也有利于公司目标方针的贯彻，并且项目成员在事业稳定性上的顾虑减少了。这种组织结构的缺点是容易引起职能经理和项目经理权力的冲突；资源共享也能引起项目之间的冲突；项目成员有多头领导。

4）小组结构

在软件项目中，通常将人员划分为若干小组（Team），每个小组负责一些任务。小组的组织结构对小组的工作效率和工作质量有很大影响。小组的结构形式可分为三类。

（1）民主分散型（Democratic Decentralized，DD）。这种软件项目小组没有固定的负责人，“任务协调人”是根据不同的任务临时指定的，随后将由协调别的任务的人取代。在实际工作中，用全体组员协商一致的方法对问题及解决问题的方法做出决策，小组成员间的通信是水平的。

（2）控制分散型（Controlled Decentralized，CD）。项目小组有一个固定的负责人，他协调特定任务的完成并指导负责子任务的若干二级管理者的工作。解决问题仍然是一项群体活动，但是可以通过小组负责人在不同的成员或成员组之间划分任务来实现解决方案。个人和成员组内部的交流是水平的，同时也存在沿着控制层次的上下级之间的通信。

（3）控制集中型（Controlled Centralized，CC）。小组负责人管理顶层问题的解决过程并负责组内协调，负责人和小组成员之间的通信是上下级式的。

CC型结构能够更快地完成任务，它最适于处理简单问题。CD型的小组比起个人来，能够产生更多、更好的解决方案，这种小组在解决复杂问题时成功的可能性更大。小组的性能与必须进行的通信量成反比，所以开发规模很大的项目时最好采用CC或CD结构的小组。小组生命周期的长短会影响小组的士气。经验表明，DD小组结构能导致较高的士气和较高的工作满意度，因此适合于生命周期较长的小组。DD小组结构最适于解决模块化程度较低的问题，因为解决这类问题需要更多的通信量。如果能够达到较高的模块化程度（人们自己独自做自己的事情），则CC或CD结构更适宜。CC和CD小组产生的缺陷比DD小组少，但是这些数据在很大程度上取决于小组采用的质量保证活动。完成同一个项目，CD结构通常需要比CC结构更多的时间，不过当需要高社交性时，CD结构是最适宜的。

历史上最早的软件项目组是CC结构，当时人们把这样的软件项目组称为主程序员组。

选择项目小组结构时，应该考虑下述七个项目因素：

（1）待解决问题的困难程度

（2）待开发程序的规模（用代码行或功能点度量）

（3）小组存在的时间（小组生命周期）

（4）问题可已被分解和模块化的程度

（5）对待开发系统的质量和可靠性的要求

（6）系统交付日期的严格程度

（7）项目所需要的交流的频繁程度

5）四种组织范型

软件项目小组的四种“组织范型”如下所示。

（1）封闭式范型：按照传统的权力层次来组织项目组（类似于CD小组）。当开发与过去已经做过的产品相似的软件时，这种项目组可以工作得很好，但是在这种封闭式范型下难以进行创新性的工作。

（2）随机式范型：松散地组织项目组，小组工作依靠小组成员发挥个人的主动性。当需要创新或技术上的突破时，用随机式范型组织起来的项目组能工作得很好。但是，当需要“有次序地执行”才能完成任务时，这样的项目组就可能陷入困境。

(3) 开放式范型：这种范型试图以一种既具有封闭式范型的控制性，又包含随机式范型的创新性的方式来组织项目组，通过大量协商和基于一致意见做出决策，项目组成员相互协作完成工作任务。用开放式范型组织起来的项目组很适于解决复杂问题，但是可能没有其他类型小组的效率高。

(4) 同步式范型：按照对问题的自然划分，组织项目组成员各自解决一些子问题，他们之间很少有主动的通信需求。

2. 项目团队建设

项目团队是一组为实现一个共同目标而协同工作的个体成员，团队工作就是团队成员为实现这一共同目标而共同努力工作。软件项目团队具有以下特点：

- 工作自主性要求高。
- 崇尚智能，杜绝权威。
- 成就动机感强，追求卓越。
- 知识创造过程的无形性。

项目团队组建完成并不等于团队建设完成，团队建设是伴随项目团队的整个生命周期的活动。项目团队建设就是项目组织通过招聘或其他方式获得项目所需要的人力资源，对他们进行必要的培训，并根据他们的技能、素质、经验、知识等进行工作安排和配备，从而构建一个项目团队。软件项目团队中常见的岗位角色包括系统分析员、系统设计员、数据库管理员、支持工程师、业务专家(用户)、测试人员等。作为软件项目计划的一部分，应在团队建设阶段制定团队成员管理计划，以描述项目团队的人员什么时候及如何加入到团队中和离开团队，详细程度因项目而异。

(1) 在对项目成员配备工作时，应根据以下原则：

- 人员配备必须要为项目目标服务。
- “以岗定员”，保证人员配备的效率，充分利用人力资源，不能以人定岗。
- 项目不同阶段所需要的人力资源种类、数量不同，要安排一定比例的临时工作人员，根据项目的需要加入或退出，以节约人力资源成本。

(2) 团队成员的培训。

通过对团队成员的培训，可以提高项目团队的综合素质、工作技能和技术水平。同时有助于提高项目成员的工作满意度，降低项目人员的流动比例和人力资源管理成本。针对项目的一次性和约束性(主要是时间和成本的制约)的特点，对于团队成员的培训主要采取短期性的、片段式的、针对性强、见效快的培训。

培训方式主要有两种。

- 岗前培训：对团队成员进行一些常识性的岗位知识和项目管理知识的培训。
- 岗上培训：主要根据开发人员的工作特点，针对操作中可能出现的实际问题，进行特别培训，多偏重于专门技术和特殊技能的培训。

(3) 冲突管理。

在项目团队管理中，冲突无时不在。软件开发工作的高压环境、沟通与知觉差异、角色混淆、责任模糊、项目中资源分配及利益格局的变化、技术上的不同观点、目标差异等都可能引起团队成员之间的冲突。从发生的层次和特征的不同，项目冲突可以分为：人际冲突、群

体或部门冲突、个人与群体或部门之间的冲突以及项目与外部环境之间的冲突。冲突管理通常有以下几种方法。

- 问题解决(Problem Solving):双方一起积极地定义问题、收集问题信息、开发并且分析解决方案,直到最后选择一个最合适的方法来解决问题。这是冲突管理中最有效、最可取的一种方法。
- 妥协(Compromising):双方协商并且都做出一定程度的让步,寻找一种能使双方都可接受的方法。
- 求同存异(Smoothing):双方都关注他们同意的观点,而避免会产生冲突的观点。
- 撤退(Withdrawal):把眼前的问题搁置起来,等以后再解决。
- 强迫(Forcing):采用一方的观点,否定另一方的观点。一般不推荐采用这种方法。

(4) 团队成员的绩效评估。

绩效评估就是工作行为的测量过程,即用过去制定的标准来比较工作绩效的记录及将绩效评估结果反馈给团队成员的过程。绩效评估的过程为:

- 建立业绩考核体系。
- 将业绩期望告知团队成员。
- 测量实际业绩。
- 比较实际业绩和标准。
- 进行矫正。

(5) 团队的激励。

激励是影响人们的内在需要或动机,从而加强、引导和维持行为的一个反复的过程,属于一种心理需要。在管理学中,激励是指管理者促进、诱导下属形成动机,并引导其行为指向特定目标的活动过程。激励机制在团队建设中十分重要,它能够调动团队成员的工作热情,增强团队士气,发挥人力资源的最大潜能。如果一个项目经理不知道如何激励团队成员,便不能胜任项目管理工作。激励理论包括马斯洛的需要层次理论、双因素理论、ERG 理论、成就需要理论、期望理论、公平理论等。激励因素是指诱导一个人努力工作的因素或手段,包括物质激励、精神激励和其他激励手段。

(6) 团队精神。

项目团队的核心是共同承诺,就是共同承担集体责任,其团队精神应该包括下述几个方面。

- 高度的相互信任
- 强烈的相互依赖
- 统一的共同目标
- 全面的互助合作
- 关系平等和积极参与
- 自我激励和自我约束

3. 项目团队管理

项目团队管理活动包括为提高团队运作水平而进行的管理和采用的措施。项目经理在管理软件项目的时候,应从管理制度、项目的目标、工作氛围、沟通等方面做工作,以保证项目的顺利进行。

1）制定良好的规章制度

一名优秀的管理者首先是一个规章制度的制定者。规章制度包括纪律条例、组织条例、财务条例、保密条例、奖惩制度等。制定了规章制度以后，应当依照制度认真执行，才能营造出良好的工作环境和氛围。

2）建立明确的共同目标

不同的团队成员由于其地位和看问题的角度不同，对项目的目标和期望值会有很大的差别。好的项目经理善于捕捉成员的不同心态，理解他们的需求，帮助他们树立共同的奋斗目标，劲往一处使，使得团队的努力形成合力。

3）营造工作氛围

项目经理为了创造一个积极进取、团结向上的工作氛围，需要做的是：奖罚分明、公正，让每个成员承担一定的压力，在软件项目问题讨论上，要民主、平等，充分调动每个成员的积极性。

4）良好的沟通

软件开发项目中，需求变更是最难控制的。在实际开发中，特别是在大型复杂系统的开发中，往往到了项目的最后阶段还可能提出需求上的变更。所以要做好大型项目的人力资源管理，就必须建立良好的沟通渠道和机制，通过沟通才能了解项目成员的想法，消除理解偏差、达成共识。

5）沟通管理

沟通管理的目标是及时并适当地创建、收集、发送、储存和处理项目的信息，基本原则是及时性、准确性、完整性、可理解性。

项目沟通方式按传播媒介的方式可划分为：书面沟通、口头沟通、非语言沟通和电子媒介；按组织系统可分为正式沟通和非正式沟通；按信息传播方向可分为上行沟通、下行沟通、平行沟通和越级沟通。

在项目沟通中，不同信息的沟通需要采取不同的沟通方式和方法，因此必须明确各种信息需求的沟通方式和方法。影响项目选择沟通方式方法的因素主要有以下几个：

- 沟通需求的紧迫程度
- 沟通方式方法的有效性
- 项目相关人员的能力和习惯
- 项目本身的规模

在项目沟通管理中，应通过制定项目沟通计划（Project Communication Plan）确定项目相关人员的信息和沟通需求：谁需要什么信息、什么时候需要、怎样获得、选择什么沟通模式、什么时候采用书面沟通、什么时候采用口头沟通等。沟通计划是整个项目计划的一个附属部分，常与组织计划紧密联系在一起，应在项目的初期阶段完成。在项目进行过程中，要根据沟通需求和计划实施的结果随时对其进行检查和修订，以保证它的持续有效性和适用性。

项目沟通计划的编制就是要根据收集的信息，先确定出项目沟通要实现的目标，然后根据项目沟通目标和确定项目沟通需求去分解得到项目沟通的任务，进一步根据项目沟通的时间要求去安排这些项目沟通任务，并确定保障项目沟通计划实施的资源和预算。

项目沟通计划书的内容除了沟通的目标、任务、时间要求、具体责任、预算与资源保障以外，一般还应该包括下列特殊内容：信息的收集和归档格式要求；信息发布格式与权限的

要求；对所发布信息的描述；更新和修订项目沟通管理计划的方法；约束条件与假设前提。

项目沟通计划主要包括以下几方面的内容：

- 描述信息收集和文件归档的结构。这一结构用于收集和保存不同类型的信息。对于不同类别的信息资料，如果缺乏及时清晰的整理和保存，就会造成信息的缺失和部门间或成员间信息的不一致。所以，有必要制定和遵循一个制度，将与项目有关的重要信息资料和文件进行建档管理。
- 项目干系人联系方式。应该有一个专用于项目所有相关人员联系方式的小册子，包括项目组成员、项目组上级领导、行政部人员、技术支持人员等系统相关人员的座机号码、手机号码、职能等，最好能有特殊人员的一些具体标注。
- 传送重要项目信息的格式。项目组的成员在准备书面和口头报告时，是否存在可以遵循的格式？是否所有的缩写词和定义都有一个列表？在不同的项目文件中它们是否需要被重复列出？建立一个项目整体统一的文件模板，是正规管理的一部分，所以必须统一各种文件模板，并提供报告编写指南。
- 用于创建信息的日程表。是否已经分配资源去创建、聚集和发送关键项目信息？项目干系人是否知道什么时候期望提交不同的信息？什么时候他们需要参加重要的会议？时间安排是否考虑到了关键项目文件的审评和批准？保证时间以建立关键项目信息和确保其质量是非常重要的。
- 获得信息的访问方法。谁能看到一个草拟的文件？每个人都能访问所有的项目文件吗？哪些信息在线保存以及哪些只保存在硬拷贝中或以其他形式保存？每个人都能检查硬拷贝中的文件吗？
- 工作汇报方式。明确表达项目组成员对项目经理或项目经理对上级和相关人员的工作汇报方式、时间和形式，比如项目组成员对项目经理通过 E-mail 发送周报；项目经理对直接客户和上级按月通过 E-mail 发送月报；紧急汇报通过电话及时沟通；每两周项目组进行一次当前工作沟通会议；每周与客户和上级进行一次口头汇报等。
- 沟通计划维护人。随着项目的推进和发展，需要更新沟通管理计划和方法。沟通计划维护人要明确在发生变化时，由谁进行修订、新的计划如何发送以及向哪些相关人员发送。

10.4.5 软件开发质量管理

软件开发质量管理层次的初步划分如下所示：

- 技术层次(数据、编程和文档)
- 方法体系层次(措施、项目和过程)
- 社会因素层次(质量环境、技术标准、业务标准和人员)

软件开发质量管理层次模型如图 10-47 所示。

1. 技术层次

1) 数据质量管理层次

在多数情况下，软件系统的最终目的是对用户关心的各类数据(信息)完成各种各样静态地或者动态地处理或管理任务，为用户创造他们所期望的和额外的价值。因此数据质量

是用户最为关心的,数据质量也反映了软件系统产品的质量。数据质量是数据抽取、数据转换、数据整合、数据仓库以及管理信息系统开发等项目中,质量控制和质量保证必须考虑的主要工作。数据质量管理可分为人工比对、程序比对、统计分析三个层次。

2)编程质量管理层次

软件系统是靠“编”出来的,为了确保软件产品的质量,就必须确保软件程序代码的质量。为了提高编程质量,应检查源代码的逻辑、属性、对象命名标准、语言代码布局等内容;代码的编译、链接、集成和构建必须得到验证和确认。编程质量管理层次可分为黑盒测试、灰盒测试、白盒测试、编译检查、编程规范、编程逻辑和编程优化。

3)文档质量管理层次

文档(包括模型)是软件开发过程中的中间成果,这些中间结果关系到软件需求的准确性和完整性、设计的合理性,对软件系统的最终结果有决定性作用。文档质量管理层次包括文档规范、文档语法、文档语义、文档逻辑、文档美学和文档优化。

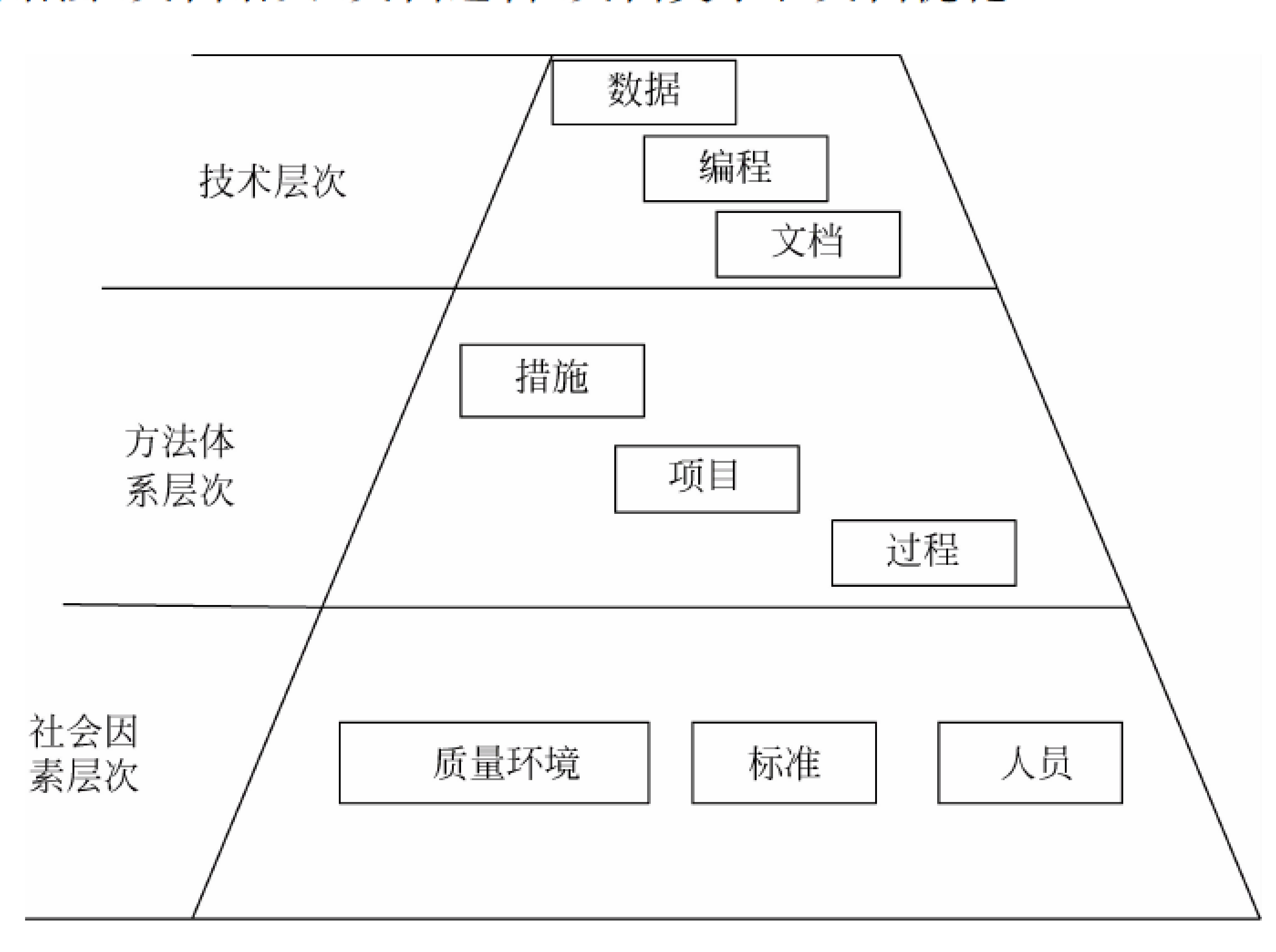


图 10-47 软件开发质量管理层次模型

2. 方法体系层次

1)措施质量管理层次

为了提高软件质量,措施质量管理是软件开发组织所采取的各种相关措施。决定软件开发项目成败的不是目标,而是措施。任何好的规章制度或计划,最终都需要具体措施才能落到实处。措施质量管理包括质量检查、质量保证、预防不合格品和完美无缺。

2)项目管理质量层次

软件开发任务一般是以项目的形式完成,项目管理质量包括组织资源、组建团队、设定目标、确定范围、确定优先级、管理风险、建立沟通机制等内容,其层次分为通用术语、通用过程、单一方法、基准比较和持续改进。一个成功的项目,不仅要满足传统的项目时间、费用和性能的三大目标以及满足客户或用户定义的质量标准,还要满足具有最少的或者双方同意的范围变更、没有干扰组织的企业文化或者价值观、没有干扰组织的日常工作进程等条件。

3）过程质量管理层次

软件生命周期的各个过程可分为三类，即基本生命周期过程、支持生命周期过程和组织生命周期过程。对于质量，强调的是对过程的重视，通过保证每个环节的工作质量，来保证最终质量，而不是通过最后的检验测试找出缺陷。它们的质量概念，不仅涵盖产品，还包括软件开发组织的整个运营过程，贯穿产业链的各个环节。确保生产出高质量的软件产品，就是要遵循一套有质量原则的软件开发过程。目前软件过程改进的主要依据是能力成熟度模型集成（CMMI）。

3. 社会因素层次

1）软件组织质量环境层次

软件组织质量环境是有关创建和管理质量环境的管理，包括质量规划、资源组织、提供相关工具等，其层次可分为质量形象、质量制度、质量战略、质量文化、企业文化、全社会质量意识。

2）标准质理管理层次

国际竞争有三个层次：第一个层次是价格和质量的竞争；第二个层次是专利技术的竞争；第三个层次是标准和制度的竞争。谁掌握了标准的使用，谁掌握了标准的制定权，谁就可以抢占先机。

标准主要包括技术标准和业务标准两大类（当然还存在其他分类，如基础标准、产品标准、质量标准、管理标准、工作标准、安全标准、术语标准等）。对标准化领域中需要协调统一的技术事项所制定的标准，被称为技术标准。技术标准包含两个方面：一是作为软件开发组织的软件行业技术标准，包括知识体系指南、过程标准、建模标准、质量管理标准、程序语言标准和数据库标准；二是软件开发服务对象所在的行业技术标准，如安全保密标准和技术性能标准。业务标准指的是软件开发服务对象所在的组织或行业制定的业务流程标准、业务数据标准等。运用统一的技术与业务标准是对于质量能够做出重大而且显著贡献的因素之一，有助于减少无效讨论，有助于不同的产品之间的兼容和衔接。标准要不断地与时俱进，因此，标准是一种动态信息。

3）人员质量管理层次

人员质量是所有工作质量的基础。要提高工作质量，就要以人为本，根本的问题是提高人的质量。人员质量就是人员素质，层次分为个人素质、团队素质、组织素质、行业素质和国民素质。

10.5 本章小结

软件过程成熟度是指一个具体的软件过程被明确地定义、管理、评价、控制和产生实效的程度。

软件能力成熟度模型（CMM）将软件组织的管理水平划分为五个级别：初始级（CMML1）、可重复级（CMML2）、定义级（CMML3）、定量管理级（CMML4）和优先级（CMML5）。分别描述了不同完善程度软件过程的不同特点，主要用于软件开发过程和能力的评价和改进，侧重于软件开发过程的管理及工程能力的提高与评估。

软件能力成熟度模型集成(CMMI),是一套融合多学科的、可扩充的产品集合,同时也是工程实践与管理方法。它能够解决现有的不同CMM的重复性和复杂性,并减少由此引起的成本,可缩短改进工程。CMMI的表达方式有阶段式和连续式两种：阶段式CMMI类似于SW-CMM,将所有的过程域按五个成熟度等级来组织,称作“CMMI成熟度等级”,从低到高依次为：初始级、已管理级、已定义级、量化管理级和优化级；连续式CMMI将PA分为过程管理、项目管理、工程以及支持,每个PA划分为六个能力能级,称做“CMMI能力等级”。CMMI的实施方法IDEAL模型是一个软件组织用于启动、规划和实现软件过程改进措施蓝图的模型,概括了建立一个成功的过程改进项目的必要步骤。

软件配置管理是识别和定义系统中配置项的过程,通过配置管理可以在生命周期中控制配置项的变更,记录并报告配置项及变更需求的状态,检验配置项的完整性和正确性。它的内容包括版本控制、变更控制和过程支持；核心是版本控制,即对系统不同版本进行标识和跟踪管理的过程。目前,软件配置管理工具已经从简单的版本控制和半自动构造系统进化到现在复杂的软件配置管理,常用的有：CCC/HAVEST、VSS、CVS和ClearCase。通常软件配置管理的流程为：人员确定与分工；项目计划阶段的管理；项目开发维护阶段的管理。

一个大型软件系统的质量应该从可靠性、易理解性、易维护性、效率等几个方面全面地进行评价。软件质量保证就是一个系统性的工作,用于提高软件交付时的水平。软件质量保证活动包括建立SQA小组、SQA活动和SQA活动的验证。软件质量保证的措施主要有基于非执行的测试、基于执行的测试和程序正确性证明。

软件项目管理是为了使软件项目能够按照预定的成本、进度、质量顺利完成,运用专门的知识、技能、工具和方法,对人员、产品、过程、项目进行分析和管理的活动。它的主要活动通常包括：软件生命周期管理、软件项目成本管理、软件项目时间管理、软件项目人力资源管理和软件开发质量管理。

习题10

1. 什么是软件过程？什么是软件过程能力？什么是软件过程成熟度？
2. 简述CMM的结构。
3. 简述阶段式CMMI的结构。
4. 什么是IDEAL模型？
5. 软件配置管理的含义是什么？
6. 软件配置管理工具的功能有哪些？
7. 简述软件配置管理流程。
8. 影响软件产品的质量因素有哪些？
9. 软件质量保证的主要任务是什么？
10. 软件质量保证的措施主要有哪些？
11. 简述软件项目管理的五个方面。
12. 简述常用的成本估算方法。

参 考 文 献

[1] 赵池龙，杨林. 实用软件工程.3版[M]. 北京：电子工业出版社，2012.
[2] 贾铁军，甘泉. 软件工程与实践[M]. 北京：清华大学出版社，2012.
[3] 殷人昆，郑人杰. 实用软件工程.3版[M]. 北京：清华大学出版社，2011.
[4] 郑人杰，马素霞. 软件工程概论[M]. 北京：机械工业出版社，2010.
[5] 张海藩. 软件工程导论.5版[M]. 北京：清华大学出版社，2009.